SURE: Shizuoka University REpository

http://ir.lib.shizuoka.ac.jp/

Title	ソフトウェア開発組織における生産技術に関する研究
Author(s)	居駒, 幹夫
Citation	
Issue Date	2011-08
URL	http://doi.org/10.14945/00007485
Version	ETD
Rights	

This document is downloaded at: 2015-11-05T23:06:38Z

静岡大学博士論文

ソフトウェア開発組織における生産技術に関する研究

2011 年 8 月

自然科学系教育部

情報科学専攻

居駒　幹夫

目次

内容梗概 1
第一章　序論 3
1.　研究の背景 3
2.　本研究の目標 5
3.　本論文のアプローチ 6
第二章　ソフトウェア生産技術の概観 9
1.　ソフトウェア生産技術 9
1.1.　ソフトウェア生産技術の必要性 9
1.2.　ソフトウェア生産技術の定義 9
1.3.　ソフトウェア生産技術の適用範囲 10
2.　過去の取り組み 11
2.1.　経営工学での取り組み 12
2.2.　ソフトウェア工学でのソフトウェア開発組織論の流れ 14
2.3.　ソフトウェア開発組織に関連する最近の話題 19
2.4.　ソフトウェア開発組織での現状の課題まとめ 22
3.　ソフトウェア生産技術のモデル化 24
4.　ソフトウェア生産技術の対象と業務機能 24
4.1.　ソフトウェア生産技術の対象 25
4.2.　ソフトウェア生産技術の業務機能 31
5.　まとめ 41
第三章　ソフトウェア生産技術の実用例 43
1.　大規模ソフトウェア開発組織のソフトウェア開発プロセス 43
1.1.　概要 43
1.2.　大規模ソフトウェア開発組織のプロセス構成 43
2.　組織的なソフトウェア開発での典型的な課題 45
3.　ソフトウェアの組織的開発の実例 46
3.1.　継続的改善の事例１：プロジェクト単位での採取データの組織化 46
3.2.　継続的改善の事例２：開発効率の導入 50
3.3.　組織化の事例　構成管理システムの組織共通化 56
3.4.　規律・統制の事例　組織知の展開事例 59
4.　本章のまとめ 61
第四章　大規模ソフトウェア開発組織での Validation モデルを使った回転率の適用 63
1.　はじめに 63
2.　関連分野の動向及び課題 64

2.1　日本のソフトウェア工場アプローチとその課題 64
2.2　反復型の開発プロセスにおける指標の課題 64
2.3　仕掛かりを用いた回転率指標およびその課題 66
3.　アプローチ 68
3.1 ソフトウェアの Validation モデルとソフトウェア開発回転率 68
3.2　ソフトウェア開発回転率における単位の設定方法 72
4.　大規模ソフトウェア開発組織での適用実例 73
4.1　組織概要及び適用対象 74
4.2　Validation モデルの具体的適用方法 74
4.3　Validation モデルの適用対象 75
4.4　ソフトウェア開発回転率及び生産性の推移 76
5.　考察 78
5.1　Validation モデルの特長，考慮事項 78
5.2　適用結果の考察 79
6.　おわりに 81
第五章　コラボレーション活性化と企業活動の適正化を 両立させる企業情報システムモデル 85
1.　はじめに 85
2.　現状の課題 86
3.　コラボレーション基盤管理の考え方 87
4.　コラボレーション基盤の分割統治 88
4.1　コラボレーション基盤の層ごとの分割 88
4.2　事業の段階ごとにコラボレーション基盤を分割 90
4.3　ＯＦＦコラボレーションモデル 93
5.　適用事例及び評価 93
5.1　コラボレーション基盤の構成，使用法 94
5.2　提案モデルの適用結果 96
6.おわりに 99
第六章　結論 101
1.　成果 101
2.　今後の課題 102
謝辞 103
文献目録 104
図表目次 111
索引 114

内容梗概

本研究は，ソフトウェア開発組織を「ソフトウェア生産システム」とモデル化し，多数のソフトウェア開発プロジェクトを効率的に運用し，高品質なソフトウェアを開発し続けるシステムの構築，運用，改善方法について述べる．

本研究の適用範囲は「組織的なソフトウェア開発」である．ソフトウェア工学では，すべてのソフトウェア開発をプロジェクトとしてモデル化している．このモデルでは，ソフトウェアの開発を開始時に体制の編成を行い，その後，計画，実行，制御を経て，終了するというプロセスを経る．一方，現実の大規模ソフトウェア開発のほとんどは，法人，事業所，事業所内の部などの固定的な組織によって開発されている．このような組織では，一つ一つのソフトウェア開発ごとに開発体制も含めてダイナミックに編成する場合よりも，固定的なソフトウェア開発部署がその業務機能として定常的に同種のソフトウェアを開発する場合が多い．本研究は，このようなソフトウェア開発組織において，継続的かつ効率的に高品質なソフトウェアを開発する方法を適用範囲とする．

本研究において二点の大目標「組織的なソフトウェア開発方法の枠組みの提案」と，「多様化する環境に対応した組織的な開発方法」を立てた．

一点目の目標は，組織的なソフトウェア開発方法の枠組みの提案である．本研究では，従来のソフトウェア工学が提供する知識だけでなく，経営工学の知識を統合する．この目標のため，まず，ソフトウェア開発組織を「ソフトウェア生産システム」，すなわち，複数のソフトウェア開発プロジェクトを定常的に運用し永続的にソフトウェアを開発し続けるシステムとしてモデル化する．続いて，「ソフトウェア生産技術」を，ソフトウェア開発組織の持つ限られたリソースを総合，最適化してソフトウェア開発プロセスを構築，改善する技法の体系と定義する．ここで，限られたリソースとして，組織の持つ「人」「プロセス」，「成果物」，「開発支援」，「知識」をソフトウェア生産技術の対象として定義する．また，ソフトウェア生産技術の業務機能として「改善」，「組織化」，「規律・統制」を定義する．さらに，定義したソフトウェア生産技術の対象と業務機能が，実際の大規模ソフトウェア開発組織で，どのように使用できるかを示し，モデルが現実の組織において活用可能であることを示す．

二点目の目標は，昨今の多様化する環境に対応した組織的な開発方法の提案である．昨今，ソフトウェア開発分野では，開発技術，開発プロセス，開発環境，個々の開発における要件等のどれをみても変化が激しく，開発のスピードアップが求められている．さらに，開発するソフトウェアの知的財産権が複雑になるにつれ，組織としての効率向上と権利保全を両立することが困難になりつつある．本研究では，これら最近のソフトウェア生産技術の課題に対応した研究成果を二つ紹介する．

一つ目の研究成果は，他産業で広く活用されている効率計測メトリクスである回転率の

ソフトウェア開発への適用方法の確立である．現状，ソフトウェア開発の効率メトリクスとして使用されている生産性は，ソフトウェア開発のスピードアップという目標に対応できない場合がある．また，組織の俊敏さを計測するメトリクスとして多くの産業分野で活用されている回転率のソフトウェア開発への具体的な適用方法や，大規模な適用結果は示されていなかった．本研究では，ソフトウェア開発組織で回転率を計測できる項目の条件及び，その計測方法を明確にし，ソフトウェア開発回転率として提案する．ソフトウェア開発回転率は，多様なソフトウェア開発プロセスモデルに従ったソフトウェア開発プロジェクトを多数持つようなソフトウェア開発組織でも適用可能で大規模ソフトウェア開発組織での実用結果を通し，提案したメトリクスが実際のソフトウェア開発で使用可能での有効性を検証する．

二つ目の研究成果は，ソフトウェア開発組織内での情報の共有と，多様な知的財産権に由来する情報保全への要求を両立させる情報管理モデルである．現在，ソフトウェア開発組織内の情報交換，知識形成の手段として電子メールや Web ベースのシステムが主流になっている．さらに電子的なコラボレーションを可能にする手段，Wiki，Blog，SNS 等のサービスを組織内で活用したいという要求が高まっている．一方，ソフトウェア開発においては，知的財産権の保護，セキュリティ確保等の各種規制に対応する目的で電子的なコラボレーション手段を抑制することも求められている．本研究では，この相反する要求を両立させることを目的とし，組織内のコラボレーションシステムの構成，活用される事業の段階ごとにコラボレーションに対する要件を明確にし，それらの要件を満足させる管理モデル，ＯＦＦコラボレーションモデルを提案する．また，提案したモデルを大規模ソフトウェア開発組織で適用し，組織内のコラボレーションシステムに求められる要件を満たすことを確認する．

最後に，本研究で得られた成果をまとめ，ソフトウェア開発組織を主眼においたソフトウェア生産技術の有効性および今後の展望を示す．

第一章　序論

1. 研究の背景

ソフトウェア工学では，すべてのソフトウェア開発は，それぞれ固有の目的，固有の体制，固有のスケジュールで行われる典型的な「プロジェクト」であると言われている [1]. プロジェクトとしてのソフトウェア開発は，PMBOK [2]の言うように，その開始時にプロジェクト体制の編成等のプロジェクト初期化が必要で，その後，計画，実行・制御を経て，プロジェクトの終了となる．一方，現実の大規模ソフトウェア開発のほとんどは，法人や，事業所，事業所内の部などの固定的な組織によって開発されている．このような組織では，一つ一つのソフトウェア開発ごとに，開発体制も含めてダイナミックに編成するということは稀で，固定的なソフトウェア開発組織がその業務機能として定常的に同種のソフトウェアを開発する場合が多い．組織的にソフトウェアを開発する理由としては，一つのソフトウェアはそのライフサイクルで複数のプロジェクトを経るため固定的な組織で開発したほうが効率の良いこと，また，組織的にソフトウェア開発に関する知識を創造，蓄積することによって他社との差別化が可能なことが挙げられる．

すなわち，ソフトウェアの開発に必要な知識を考えたときに，ソフトウェア開発をプロジェクトとして毎回編成から終了までを繰り返すような従来の動的モデルとともに，ソフトウェア開発組織を主眼において，その業務機能としてソフトウェア開発を実行するような静的な組織的なモデルもソフトウェア工学に導入すべきだと考える．

ソフトウェア開発部門を多く持つ大規模なソフトウェア開発組織の場合，その組織の持つ人，金，開発環境などのリソースを使って，いかに価値のあるソフトウェアを多く，かつ継続的に作り出すかが経営的な課題となる．例えば，次の項目は大規模ソフトウェア開発組織の日常の課題であるが，ソフトウェア工学が提供する知識だけでは十分に解決できない．

- 組織のもつ限られたリソースを使って複数のソフトウェア開発プロジェクトのスケジュールをどのように適正に配置するか
- 優秀なソフトウェア開発者をどのように各プロジェクトの各工程に割り当てるか
- 高価なソフトウェア開発支援ツールを組織内でどのように活用するか
- あるプロジェクトで得た知識をいかに組織全体で共有するか

このような課題に対応しては，システム工学的な知見を企業に適用する，すなわち，経営工学の成果をソフトウェア開発組織に適用することが必要だと筆者は考える．現状のソフトウェア工学でも，ソフトウェア開発組織に関する研究は少なくない．また，ソフトウェア工学のソフトウェア開発組織論の多くは，経営工学の影響を受け続けている．この分

野でのソフトウェア工学の成果, 例えば, 1970～1990 年代の日本のソフトウェア工場 [3], Basiliの経験工場 [4], Humphreyのプロセス成熟度モデル [5]は，その根幹にハードウェア生産や開発の方法論で蓄積された知識がある．さらには，2000 年代に広く普及しつつある，アジャイルソフトウェア開発手法 [6]の多くも経営工学での開発や生産システム論の影響を受けている．Poppendieckのリーン開発手法 [7]は，トヨタ生産方式 [8]をソフトウェア開発に当てはめた開発方式であり，Sutherland等のScrum [9]の端緒は竹内・野中の新製品開発に関する論文 [10]である．また，BeckのeXtreme Programming(XP) [11]に見られるYAGNI(You Aren't Gonna Need It:今必要なことだけ行う)というプラクティスは「必要数がオールマイティ」というトヨタ生産方式のかんばん方式の影響を受けており，ソースの共同所有という考えもトヨタ生産方式の自働化をソフトウェア開発にあてはめた結果とみなせる．もっともBeckは，経営工学の創始者といわれるTaylorの方法論に対するアンチテーゼとしてXPを生み出したと言っている [1]．経営工学を教師とみるか反面教師とみるかの違いはあるが，ソフトウェア工学での組織論は，過去も現在も経営工学での成果と密接に関わりがあるといっても過言ではない．

一方，経営工学での知識がそのままソフトウェア開発組織の課題に対応するわけではない．ハードウェア開発組織の中で発展してきた経営工学や生産技術が提供する具体的な技法の中には，そのままではソフトウェア開発組織に適用できないものも多く（例えば，サプライチェーン，タグチメソッド等），適用可能な技法（例えば，統計的な品質管理）でもソフトウェア開発向けにカスタマイズが必要な場合が大部分である．

ここで，開発組織の観点からソフトウェアとハードウェアの違いについて考える．ソフトウェア工学の立場から，Humphrey [12]はソフトウェアとハードウェアの違いを以下のように述べている（著者要約）．

・一般的にソフトウェアはハードウェアよりも複雑である
・ソフトウェアはハードウェアと比較して容易に変更ができるように見える
・ソフトウェアの改修コストが低いため，製造工程にリリースするという考え方がない
・ソフトウェアに関する訓練は自然科学に基づいていない
・ソフトウェアはシステムを統合するための要素であり，複雑さを増加させると，後で変更の対象になることが多い
・ソフトウェアは外から良く見えるため，もっとも要求変更を受け止める位置づけにあり，ユーザーの不満の対象となる
・経験豊富な管理者や上層部は少ない

一方，経営学，ビジネスの立場から，Cusumano [13]はハードウェアビジネスと対比してソフトウェアビジネスの特徴を以下のように述べている（著者要約）．

[1] 2002 年(株)日立製作所ソフトウェア事業部での講演後のヒアリング

・一つのコピーを作る製造コストと 100 万のコピーを作る製造コストがほぼ同じで済むビジネスである
・製品売上げに対するマージンが 99%に達する業界である
・もっとも生産性が高い従業員ともっとも生産性の低い従業員で 10 倍～20 倍の格差がある
・プロジェクトの 20%を時間通りに成し遂げると「ベストプラクティス」と見なされる状況が許容される
・製品の開発者が自らを科学者や技術者というよりも芸術家と思っていて，移り気な気質で仕事をしていても仕方がないと会社が認められる
・10 年か 20 年前のだれかが行った製品決定に縛られて変更が効かなくなり，ユーザーが特定メーカーに「ロックイン」されてしまう

ソフトウェア，ハードウェアの違いの説明の仕方にも，ソフトウェア開発側，経営側で違いが見えるのは興味深い． Humphrey は，ソフトウェア/ハードウェアを「開発する方法」に着目した差異に焦点を当てている．これに対して，Cusumano は組織の業務において「組織の持つリソースの活用方法」にどのような差異があるかに着目している．Humphrey の言うソフトウェアの特徴に対応するのがこれまでのソフトウェア工学の組織論であるとすると，それに加えて，Cusumano のいうような組織のリソースに着目したソフトウェア工学の組織論も必要である．また，組織の持つリソースという観点での組織論の確立には，従来のソフトウェア工学が提供するソフトウェア開発組織の知識に加えて，ソフトウェア向きにカスタマイズした経営工学的な知識が必要であると筆者は考えた．

一方，大規模ソフトウェア開発組織に関する様々な課題に対応するため，多くの組織には，ソフトウェア工学および経営工学的な知識を組織に根付かせ，組織内の各プロジェクト，や開発者に展開する部署が存在する．日本では，「ソフトウェア生産技術部」と名づけられることの多いこれらの部署は，組織内のソフトウェア開発部署に対する開発支援を行う他，その組織全体のソフトウェア開発能力を把握し，そのプロセスを継続的に改善していく使命を負っている．この部署は，名が示す通り，ハードウェア製造組織における生産技術部署に対応した組織である．しかし，「ソフトウェア生産技術」とは何かという明確な定義は，規格や学会的な共通認識は無い．

2. 本研究の目標

前節で述べた課題を踏まえ，本研究は，次の二点を大目標とする．

- 組織的なソフトウェア開発方法の枠組みの提案
- 多様化する環境に対応した組織的な開発方法

一点目の目標は，組織的なソフトウェア開発方法の枠組みの提案である．組織の持つ限られたリソースを活用し，どのように多数のソフトウェア開発プロジェクトを効率的に運用するか，過去のソフトウェア開発プロジェクトで蓄積されたノウハウをどのように活用し，どのように組織知とするかといった経営工学での成果を従来のソフトウェア工学の成果と統合し，体系化することを目指す．

二点目の目標は，昨今の多様化する環境に対応した組織的な開発方法の提案である．ソフトウェア開発は現在，開発技術，開発プロセス，開発環境，個々の開発における要件等のどれをみても変化が激しく，開発のスピードアップが求められている．さらに，開発するソフトウェアの知的財産権が複雑になるにつれ，組織としての効率向上と権利保全を両立することが困難になりつつある．これらの最近のソフトウェア生産技術の課題への対応は多方面からの対策が必要であるが，本研究では，第一の目標で確立したソフトウェア生産技術の枠組みでこれらの課題に対して部分的に対応可能なことを明確にする．

3. 本論文のアプローチ

前章で述べた大目標を達成するため，本研究は，ソフトウェア開発を行う組織を「ソフトウェア生産システム」とみなせることを仮定し，ソフトウェア生産システムの持つ限られたリソースを定量的かつ方法論的なアプローチで総合，最適化する技法の体系を「ソフトウェア生産技術」とする．このソフトウェア生産技術を活用することによって本研究の目標が達成できることを以下のように示していく．

次章，第二章ではソフトウェアの生産技術の形式知化を目指す．まず，ソフトウェアの生産技術を定義する．続いて，その起源，発展を 20 世紀初頭に始まる経営工学の科学的管理法まで遡って概観し，これまでの成果と現状の課題を示す．さらに，ソフトウェア生産技術を「ソフトウェア生産技術の対象」，「ソフトウェア生産技術の業務機能」の二つに分けてモデル化する．ソフトウェア生産技術の対象として「人」，「プロセス」，「成果物」，「開発支援」，「知識」を説明し，ソフトウェア生産技術の業務機能として，「改善」，「組織化」，「規律・統制」を説明する．

第三章は，第二章で示したソフトウェア生産技術の業務機能についてその実例を示し，実際に活用可能であることを示す．具体的には，筆者の勤務している(株)日立製作所ソフトウェア事業部の過去の生産技術活動において，どのように「改善」，「組織化」，「規律・統制」を行ってきたのかを示し，本研究の第一の目標である「組織的なソフトウェア開発方法の枠組み」が現実に機能することを示す．

第四章，第五章では，本研究の第二の目標である昨今の多様化する環境に対応した組織的な開発方法についての研究成果を述べる．

第四章では，1990 年代以降ソフトウェア市場における競争激化に伴い，多様化したソフトウェア開発環境に対応し，かつ，ソフトウェア開発組織に求められるようになってきた，

俊敏な開発をどのように組織的に計測するかについて述べる．従来の生産性，すなわち成果量÷コストに加え，組織の俊敏さを計測するメトリクスとして多くの産業分野で広く活用されている回転率をソフトウェア開発においても導入できることを示し，このソフトウェア開発回転率メトリクスが，多様なプロセスモデルに従ったソフトウェア開発プロジェクトを多数持つようなソフトウェア開発組織でも適用可能であることを示す．また，組織全体での適用結果を示し，その有効性を検証する．

第五章では，昨今，ソフトウェア製品開発組織に求められるようになってきた組織的な知識の形成と，知的財産権への対応を両立させる情報管理モデルを提案する．多数のソフトウェア開発プロジェクトを常時運用する組織は，複数プロジェクトを跨り，組織全体で知識を積極的に共有したほうが良い．一方ソフトウェア開発においては，知的財産権の問題などにより，同一組織内であっても適切な情報管理が必要な場面もある．この章では，組織としての情報の共有と，多様な知的財産権に由来する情報保全への要求を両立させる情報管理モデル，ＯＦＦコラボレーションモデル (Open, Flexible, and Formal collaboration model)を提案し，適切な知識共有と分離を実現する知識共有インフラと，製品開発の各フェーズでどのように使い分けるかを示す．

最後に，本研究の成果をまとめ，本研究の目標が達成できたことを示す．また，今後の課題およびソフトウェア生産技術の今後の展望を示す．

第二章　ソフトウェア生産技術の概観

本章では，ソフトウェア生産技術の形式知化の枠組みを提案する．まず，第一節で本研究のテーマであるソフトウェア生産技術の必要性を説明し，その定義，適用範囲を示す．第二節では，ソフトウェア生産技術の起源を経営工学の科学的管理法まで遡って概説し，これまでの成果および現状の課題をまとめる．最後に，第三節でソフトウェア開発組織におけるソフトウェア生産技術のモデルを示し，その中の「ソフトウェア生産技術の対象」，「ソフトウェア生産技術の業務機能」の内容を述べる．これらは，特定のソフトウェア開発組織とは独立に活用可能なモデルとして提案し，ソフトウェア業界全体で活用可能な形式知とすることを目指す．

1.　　ソフトウェア生産技術

1.1.　ソフトウェア生産技術の必要性

現実のソフトウェア開発組織を運用する立場では，ソフトウェア工学の知見，経営工学的な知見を分離して適用することはできない．なぜならば，どの知見であっても実際のプロジェクトで適用するためには，コストが必要であり，組織が投資できる有限のコストの中で，いろいろなタイプの投資をより効果的に配分することが必要だからである．例えば，次のような組織の施策は工学の観点では違う分野の施策である．

- 再利用可能な部品を購入
- 組織的な形式知を管理するデータベース作成
- 組織的な定量的な品質管理

しかし，現実の組織では，コストと効果いう観点で，同じ俎上で実施するか否かを議論されなければならない．序論でも述べたとおり，このような様々な分野の組織的な課題に対処するため，「生産技術部」と名付けられた部署を持つソフトウェア開発組織も多い．しかし，これらの部署が持つ知識やスキルはその開発組織に閉じた暗黙知にとどまっており，その定義や，モデルは明確でない．従って，本研究では，まず本節で「ソフトウェア生産技術」という用語を定義し，どのような適用範囲を持つのかを明確にする．さらに，三節でソフトウェア生産技術を形式知化する枠組みを示す．

1.2.　ソフトウェア生産技術の定義

本研究では，ソフトウェア生産技術（Software Industrial Engineering）を以下のように定義する．

ソフトウェア開発組織の持つ有限なリソースを工学的手段，すなわち定量的かつ方法論的なアプローチで総合，最適化することにより，その組織のソフトウェア開発プロセスを構築，改善する技法の体系

ソフトウェア開発組織の組織的な課題に対応するため，組織の持つソフトウェア開発プロセスの構築・改善を行う技法の体系とした．さらに，経営工学で蓄積された知見をその構築，改善に活用できるようにするため，組織の持つ「有限なリソース」を工学的な手段で活用するという手段を定義した．すなわち，従来のソフトウェア工学での知見に加えて，経営工学での成果を実際のソフトウェア開発を行う組織に活用するための技法の体系としてソフトウェア生産技術を定義した．

生産技術と混用されることの多い用語として生産管理という用語がある．ハードウェア製造において，生産管理とは生産システムの中で処理される部品や製品の品質(Quality)・コスト(Cost)・納期(Delivery)（以下，QCD と略す）を管理することである．一方，生産技術とは生産システム自体を構築する業務機能をいう．この定義をソフトウェア開発にあてはめると，生産管理は各ソフトウェア開発における QCD を管理すること，すなわちソフトウェアプロジェクト管理に相当する．一方，生産技術は複数のソフトウェア開発プロジェクトを効果的かつ効率的に実行するような組織的なシステムを構築し，組織全体で効率化，改善，統制することに相当する．従って，ハードウェアと同様に，ソフトウェアを開発する組織でも，例えば，プロジェクト管理部という部署の業務機能（個別のプロジェクトの実行を管理支援）と，生産技術部という部署の業務機能（各プロジェクトの実行する組織的なシステムの構築）は異なる．

1.3. ソフトウェア生産技術の適用範囲

ソフトウェア生産技術の適用範囲を図 2-1 により説明する．

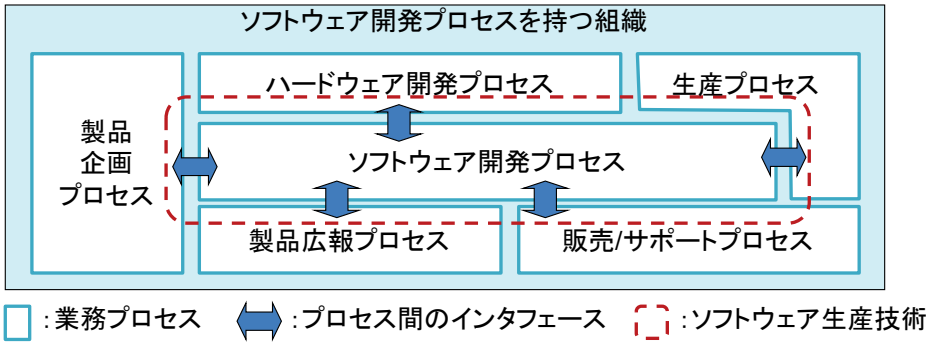

図 2-1 ソフトウェア生産技術のスコープ

ソフトウェア生産技術は，以下二つの条件をともに満たす組織に適用可能である．

- 業務プロセスとしてソフトウェア開発プロセスを持つこと
  ハードウェア製品を開発する組織であっても，その製品に組み込まれるようなソフトウェアの開発プロセスを持つ組織はソフトウェア生産技術を適用可能である．
- 組織として定常的にソフトウェアを開発していること
  ソフトウェア開発プロジェクトが間欠的に実行されるような組織ではなく，定常的に複数のソフトウェア開発プロジェクトが組織のもつ人や物のリソースを共有しながら実行されるような組織でソフトウェア生産技術を適用可能である．

通常，ソフトウェア開発プロセスを持っている組織であっても，ソフトウェア開発以外の多くの業務プロセスを持っている．例えば，どのようなソフトウェアを開発するか（製品企画プロセス），どのようにソフトウェアを広報するか（製品広報プロセス），関連するハードウェアの開発（ハードウェア開発プロセス）といった他プロセスはソフトウェア生産技術の適用範囲外である．しかし，ソフトウェア開発プロセスとこれら他プロセスとのインタフェース部分は対象とする．例えば，ソフトウェア開発プロセスの入力となる他のプロセスの出力（製品企画プロセスからの要求一覧，ソフトウェア開発の前提になるハードウェア仕様書等を入力する部分）や，他プロセスの入力になる，ソフトウェア開発プロセスの出力（マニュアル，出荷マスター媒体の作成する部分）はソフトウェア生産技術の対象とする．

なお，従来のソフトウェア工学ではソフトウェア開発は，量産ハードウェアのような生産プロセスとは無関係という理解がある．しかし，実際の大規模ソフトウェア開発組織，特にパッケージ型のソフトウェアを開発する組織では，開発したソフトウェアを一定の形式の媒体として生成して出荷するような生産プロセスがある．ここでは，ソフトウェア開発者にいかにソフトウェアの出荷形式や出荷作業を意識させないようなインタフェースを構築するかが重要となる．従って，この生産プロセスとのインタフェース部分もソフトウェア生産技術のタスクであり，その適用範囲とする．

2. 過去の取り組み

本節では，ソフトウェア生産技術に関連する過去の経営工学，ソフトウェア工学の取り組み，日本のソフトウェア開発組織での生産技術を振りかえる．ここでの説明は，過去の研究のソフトウェア生産技術に関連する側面のみに焦点を当てる．従って，各研究の詳細は参考文献を参照していただきたい．すでに，経営工学の知識又はソフトウェア工学の組織的な知識を持った読者は本節の該当部分を読み飛ばしていただいて構わない．

ソフトウェア工学は，1968 年にドイツで開催された NATO 会議の表題として初めて使用された [14]．しかし，ソフトウェアの組織的な開発は，それ以前から多く存在していた．例えば，米国では，1960 年代の IBM 社のシステム 360 の開発 [15]，1969 年に月への有人

宇宙飛行を実現したNASAのアポロ計画 [16]などの大規模ソフトウェア開発を含む大プロジェクトが多くある．日本においても，1960年に運用開始した日本国有鉄道（国鉄，現JRグループ）の座席指定券予約・発券用に開発されたコンピュータ・システムのMARS [17]や，1965年に稼働した三井銀行の第一次オンラインシステム等の大規模プロジェクト [18]があった．これらの大規模プロジェクトでのソフトウェア開発は，ソフトウェア工学という名前を用いずに実行されたが，実際にそれらのプロジェクトが，工学的な手法を全く用いなかったわけではなく，ハードウェア製品開発，製造で用いられた各種経営工学的な手法を使用，改善してソフトウェア開発にも用いていた．

本節では，ソフトウェア生産技術に影響を与えている，経営工学でのハードウェア生産手法を科学的管理法まで遡って説明する．

2.1. 経営工学での取り組み

（１）生産工学に特化した経営工学

第二次世界大戦前の経営工学の定義は，「定められた時間に最適な原価で生産を達成するために，人・設備・機械を利用し，協働する技法および科学」 [19]である．近代の組織的な生産技術は，20世紀初頭のFrederick W. Taylor [20]の科学的管理法（またの名をテーラリズム）に始まる．Taylorは，科学的な方法を量産品の製造工場に持ち込むことによって，高品質な製品を高効率に製造できることを示した．すなわち，Taylorは生産工程での標準的な仕事量を，技術者が科学的な方法で設定し，それから算出した単位期間の仕事量を用いて，労働者が計画的に生産活動を行うことにより，その組織の生産を上げた．また，Gilbreth [21]は，熟練工の仕事の手順を観察することにより，最適な仕事の手順を標準化することにより生産性を改善した．さらに，Shewhart [21]は数理統計による統計的品質管理をハードウェア製造に持ち込み，数学的な裏づけのある管理方法で製品の品質を向上させた．

この時点での経営工学はハードウェア製品の大量生産に特化した手法，すなわち，繰り返し作業の効率化が主であり毎回違うものを開発するソフトウェア開発とは異なる．また，どのように継続的に改善を推進するかという長期的な視点による改善プロセスも示されていない．序論で述べたとおり，ソフトウェア工学ではテーラリズムに対する批判が多数ある．ソフトウェア生産技術の立場でTaylorの科学的管理法などを評価する場合，テーラリズムの作業者の管理としての側面と，組織での定量的な測定管理といった側面を分けて考える必要がある．Taylorの「（日給制の最大の障害は労働者の）仕事ぶりのおそいこと，すなわち怠業またはいわゆる足踏みといわれているものである」 [20]という例に代表される性悪説に基づく作業者管理論は，21世紀の現在では，ハード，ソフトを問わず非現実的な理論である．一方，組織的に何らかの科学的な裏付けのある基準を設け，それに従って，プロジェクトや，組織全体の効率を把握しようというテーラリズムのもう一つの側面，組織での定量的な測定管理は，ハードウェア生産，ソフトウェア開発の方法論として現在で

も生きている．

（２）実践的システム工学としての経営工学

第二次世界大戦における軍需生産の能率増進を起源とするオペレーションズリサーチ（OR）等のシステム工学的な知識が経営工学に組み込まれ，その対象範囲もハードウェアの製造プロセスだけに限定されなくなってきた．システム工学の手法のうち PERT/CPM [22]といった，プロジェクト内のタスクを最適に割り当てる手法は，現在のソフトウェア開発のプロジェクト計画で一般的な手法になっている．また，Little のリトルの法則 [23]をソフトウェア開発に取り入れた手法は本研究の第四章で詳述する．

さらに，Deming，Feigenbaum [24]，石川 [25]らによる品質管理の進展により，労働者自身も組織のもつ知識の源だと見なされるようになり，成果物の品質管理だけでなく，生産過程の品質管理も重要となってきた．とくに，デミングサイクルの名前で良く知られる PDCA (Plan – Do – Check – Act)サイクルは，ソフトウェア開発組織が継続的にその製品，プロセス，組織そのものを改善するときの基本的なフレームワークとなっている．

このような経緯で，現在での経営工学の定義 [19]は，「工学のうちで，人・材料・設備の統合されたシステムの設計・改善・設置をなすことを対象とするもの」となっている．この定義は，ハードウェアの生産だけにとどまらず，ハードウェア製品の開発や，ソフトウェア開発に対しても当てはまめることが可能な定義である．

（３）部分最適から全体最適を目指す現在の経営工学

前項で示したように，システム工学や統計学の技法を企業の生産現場に取り入れることにより，生産システムの効率や，生産物の不良は低減した．しかし，生産システムをもつ企業全体の目標（例えば，利益，顧客満足，社会貢献）を達成するための手段としては十分でない場合がある．例えば，作業者の効率や機械の稼働率を向上させて原価低減を追求することが，かえって，仕掛りやリードタイムの増加につながり，全体システムの目標，例えばスループットや顧客満足から外れてしまう場合があった．現在の経営工学での手法，トヨタ生産システム [8]や，TOC [26]では，従来の部分的な最適化から，全体システムとしての最適化．自組織のシステムだけでなく，そのシステムが必要とする供給者のシステムまで含めた全体的なサプライチェーンでの最適化を考える．また，部分的な観点での QCD の向上ではなく，全体システムの立場で（たとえ一部のサブシステムの部分効率は下げても）全体の目標に合致したシステム構築することを目指している．

序論でも述べたように現代的な経営工学は，ソフトウェア工学，特にアジャイルソフトウェア開発手法に大きな影響を与えている．Poppendieck のリーン開発手法 [27] [7]は，トヨタ生産方式 [8]をソフトウェア開発に適用した手法であり，Beck の eXtreme Programming(XP) [11]に見られる YAGNI(You Aren't Gonna Need It:今必要なことだけ行う)というプラクティスや，ソースの共同所有という考えもそれぞれ，トヨタ生産方式のか

んばん方式や自働化をソフトウェア開発にあてはめた結果とみなせる．

2.2. ソフトウェア工学でのソフトウェア開発組織論の流れ

ソフトウェア工学の知識を，その知識を必要とするものを基点に分類できる．図 2- 2 はソフトウェア開発者個人が必要とする知識，ソフトウェア開発プロジェクト運用に必要な知識，ソフトウェア開発組織運用に必要な知識に分類した例である．例えば，構造化プログラミングの知識は，ソフトウェア技術者やソフトウェア開発プロジェクトの計画などに使う知識としては有効であるが，ソフトウェア開発組織の運用に必要な知識ではない．本項では，この分類のうち，ソフトウェア生産技術に大きく関わる大規模ソフトウェアプロジェクトや，ソフトウェア開発組織の定量化や継続的改善に関係する知識を中心にこれまでのソフトウェア工学の取り組みを概観し，関連する研究について解説する．

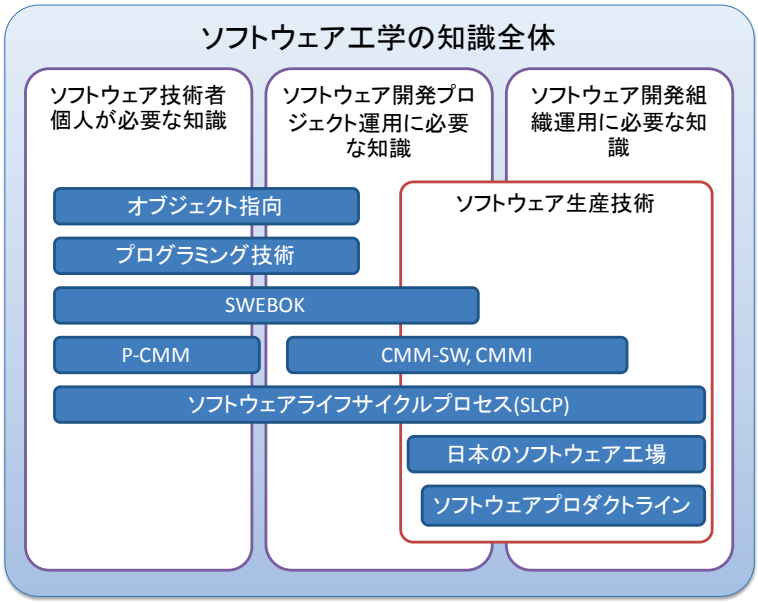

図 2- 2 ソフトウェア工学の知識の分類例

（1）ソフトウェア開発組織としての定量化の試み

1960 年代の IBM 社のシステム 360 用のソフトウェア（OS/360）の開発を通して得られた知見を述べた Brooks の著作「人月の神話」 [15]は，ソフトウェア開発の定量化が主題では無いが，1975 年に出版後のソフトウェア工学に大きな影響を与えた．例えば，次のようなソフトウェアの特性は，2000 年代のアジャイル手法にも多く引用されるほど有名になっている．

- ソフトウェアの開発にはハードウェアの製造での「人月」という単位が通用しない場合があること

- 同じソフトウェアを書くにも個人差が大きいこと
- ドキュメントも含めた組織的な開発ではコストが数倍違うこと

ただ, Brooks が示したソフトウェア開発の性質が, 本質的な性質で不可避なのか, Brooks の経験したプロジェクトの属性に依存していたのかは吟味する必要がある．すなわち，Brooks のプロジェクトでは定量的に管理できなかった問題でも，組織的な取り組みによって，管理可能になる可能性はある．実際に，1970 年以降の IBM 社の取り組み，例えば，Fagan のフォーマルインスペクション [28]，Mills のクリーンルームソフトウェア開発 [29] [30], Albrecht のファンクションポイント [31] [32], Humphrey のプロセス成熟度 [5], IBM 社が現在も全社的に推進している製品開発体系（IPD: Integrated Product Development） [33] [34]を見ても，Brooks の示したソフトウェアの管理困難な特徴を組織的な取り組みによって克服する動きとみなすことも可能である．本項では，ソフトウェア工学の知識のうち開発組織の定量化の取り組みを振り返り，Brooks が示したソフトウェアの性質がどの程度定量的に扱えるようになり，どのような課題が残っているのかを示す．

1960 年代までは，汎用コンピュータ上のソフトウェア開発組織はハードウェア開発組織に付属する形態だった．1969 年，筆者の勤務する(株)日立製作所ソフトウェア事業部は，世界最初のソフトウェア開発専門の事業所として設立された [35]．さらに 1970 年代に入り，日本では，NEC(1976 年)，東芝（1977 年），富士通（1979 年），米国においても，後にユニシスに吸収された，System Development Corporation が，1976 年にソフトウェア専門の開発事業所を設立した [35]．日本のソフトウェア工場とは，松本（ [3] p1082）によれば「ソフトウェアの生産性と品質の向上に向けた，多くのソフトウェア生産組織間の技術的および管理面での連携と協力を指す概念」である ．この管理方法は，1970 年代，1980 年代には大きな成果を挙げた．すなわち，安定した開発プラットフォーム，特定の開発言語，トレーニングされた自組織のソフトウェア開発者，画一的な開発プロセスという工場的アプローチの前提条件が満足できる場合には有効な方法であった．しかし，1990 年代以降は，多様なプラットフォーム，多様な開発言語，国内外の開発拠点，プロジェクトの特性に合わせた開発プロセス，といったプロジェクトの多様性が増加し，開発プロセスを標準化することを前提にした日本のソフトウェア工場方式の管理は年々困難になってきている．

なお, EUREKA ソフトウェア工場 [36], マイクロソフト社のソフトウェア工場 [37]は，日本のソフトウェア工場とは違う概念である．欧米でのソフトウェア工場とは，コンピュータ内のツールを工員に見立てて，自動化するイメージであるのに対して，日本のソフトウェア工場とは，人間がソフトウェアを開発することを前提に，組織内のソフトウェア開発方法の標準化を徹底的に推進することによって，ソフトウェア開発に関するばらつきを最小限するような開発方法である．

1979 年，Albrecht [31]は，ソフトウェアの提供する機能に着目したソフトウェアの成果

量測定方法ファンクションポイントを提案した．ファンクションポイントは，開発する言語や，開発するソフトウェア技術者の能力に依存せずにソフトウェアの規模を定量化する方法として画期的な手法である．本手法は，機能が定形的に定義できる業務アプリケーションプログラムの分野で現在でも活用されており，ユースケースポイント [38]などの新しい見積もり手法にも影響を与えている．一方，機能の構成を定形的に定義困難なソフトウェア，例えば，ミドルウェアや組み込みソフトウェアへのファンクションポイントおよび派生した手法，例えば COSMIC 法も提案されているが，その適用は限定的である [39].

1981 年に発表された Boehm の COCOMO（Constructive Cost Model） [40]は，Boehm の TRW 社での経験を元にソフトウェア開発プロジェクトのさまざまな属性から，そのソフトウェアを開発するのに必要な工数，コスト，納期を見積もる方法を示した．例えば，もっとも簡易手法であるベーシック COCOMO において，組み込み系のソフトウェア開発プロジェクトにおけるソフトウェア開発に必要な人月はその開発行数（KLOC: kilo lines of code）をパラメタに以下のような計算式を示した.

$$\text{ソフトウェア開発に必要な人月} = 3.6(\text{KLOC})^{1.2} \qquad (2\text{-}1)$$

COCOMOはコスト管理の方法論であったが，同じアイデアでソフトウェアの品質を見積もる方法（例えばNECのSQMAT [41] [42]，日立のSQE [43] [44] [45]等）に派生した．COCOMOの示す計算式は，一企業で帰納法的に求められた算出式であり，ある組織で有効な値が算出されたとしても，他の組織でCOCOMOの示す式で有効な値が算出されるという保証は無い．実際にICSE2007 で，BoehmはCOCOMOが示したようなモデルは，1970 年代のTRW社の開発環境 [2]に依存していたことを認めている [46].

ファンクションポイントや COCOMO の普及により，多くのソフトウェア開発組織で，ソフトウェアの品質，コスト，納期等を定量化が可能にするような取り組みが試みられた．例えば，Jones [32] [47]は，ファンクションポイントを用いて，ソフトウェアの各工程に要する工数や品質の推定までできるとしているが，提示しているのは平均値（または中間値）のみで，どれだけの分散があるかは示していない．同様な試みは，独立行政法人　情報処理推進機構（IPA: Information-technology Promotion Agency, Japan）ソフトウェア・エンジニアリング・センター（SEC: Software Engineering Center）が発行しているソフトウェア開発データ白書 [48]にも見られる．この白書では，複数社のソフトウェア開発プロジェクトデータを収集し，ソフトウェア開発プロジェクトにおける生産性や品質が見積もれることを目標としている．しかし，この白書で提供しているデータをみても，分散が大きすぎて，実際のソフトウェア開発プロジェクトの見積もりに使えるとは考えられない.

[2] TRW 社の開発環境：ICSE2007 の席で Boehm は明確に述べていないが，軍事関連企業であった TRW 社が米国国防省の明確な要件定義に従ってソフトウェアを開発していたことを示す.

COCOMO の項で述べたとおり，ある開発環境に特化して COCOMO 的な定量化をすることは現在でも可能である．しかし，ある特定の環境で求めた式や，データをそのまま，他の環境で活用することはできない，また，不特定多数の環境から求めた式やデータは信頼性が低いのが現状である．

（２）ソフトウェア開発組織としての継続的改善の試み

継続的な組織改善の大きな枠組みとしては，ソフトウェア工学においても，経営工学に由来する品質管理における PDCA(Plan - Do - Check - Act)，および知識管理に由来する SECI(Socialization - Externalization - Combination – Internalization)モデル [49]のサイクルが適用されている．

しかし，具体的にどのように組織におけるソフトウェア開発を改善していくかというプロセスには大きく二つの流れがある．すなわち，定形的な改善メニューによる組織改善を行っていくアプローチと，組織の目標に応じて改善項目やメトリクスを決めて改善を行っていくアプローチである．本節では，この両者の説明および，この両者の長所を合わせたアプローチを説明する．

(a)定形的な改善メニューによる組織改善フレームワーク

1980 年代後半, Humphrey のソフトウェア開発成熟度及びそれに派生した CMM/CMMI は，具体的な改善領域やプラクティスまで示したソフトウェア開発組織のプロセス改善フレームワークを示した．

表 2-1 ソフトウェアプロセス成熟度の５段階とキープロセスエリア

	成熟度段階	組織の主な特徴	キープロセスエリア
高レベル	レベル５ 最適化する	・プロセスからの定量的な計測値のフィードバックにより、組織のプロセス改善が継続的に実施されている。	・プロセス変更管理 ・技術変更管理　・欠陥予防
	レベル４ 管理された	・ソフトウェアの成果物とプロセスの両方に対して定量的品質目標が設定されている。	・ソフトウェア品質管理 ・定量的プロセス管理
	レベル３ 定義された	・組織全体でのソフトウェアの開発と保守の標準的なプロセスが文書化されている。 ・このレベルで確立されるプロセスは、ソフトウェアのマネージャと技術要因のより効率的な活動を支援するために利用される。	・ピアレビュー　・グループ間調整 ・ソフトウェアプロダクトエンジニアリング ・ソフトウェア統合管理 ・トレーニングプログラム ・組織プロセス定義 ・組織プロセス重視
	レベル２ 反復できる	・プロジェクト管理の方針と、その方針を履行するためのプロセスが確立されている。 ・プロジェクトの計画と進捗確認が安定的に行われ、過去の成功事例を反復することが可能である。	・ソフトウェア構成管理 ・ソフトウェア品質保証 ・ソフトウェア外注管理 ・ソフトウェア進捗管理 ・ソフトウェアプロジェクト計画 ・要件管理
低レベル	レベル１ 初期	・ソフトウェアの開発作業は場当たり的 ・プロジェクトの成功はほとんど個人の能力	なし

Humphreyの著書「ソフトウェアプロセス成熟度の改善」 [5]は，6 ステップの改善サイクルといった組織の継続的な改善にも多くのページを費やしている．また，CMMIを提供しているカーネギーメロン大学のソフトウェアエンジニアリング研究所（SEI：Software Engineering Institute）では，一般的なプロセスの継続的改善モデルとしてIDEAL（Initiating, Diagnosing, Establishing, Acting and Learning[3]）モデル [50]を提案している．しかし，1980 年代での平均的なソフトウェア開発組織のプロセス成熟度が低かったせいもあり，CMM/CMMIでは，事業固有の継続的改善をするための前提条件として，プロセス成熟度の低いソフトウェア開発組織が共通に持つべきプロセス領域の改善モデルを示している．また，段階的にプロセス成熟度を上げるために，「反復可能」「定義された」「管理された」といった各レベルを設け，各領域，各段階におけるベストプラクティスを示した．これらのレベルを満足した後に，「最適化し続ける」という最終的なレベルを置いた（表 2-1）．

CMM/CMMIで示されたプロセス領域は，CMMI自体が述べているように，実施する組織のプロセスと一対一に対応しているわけではない[4]．実体としては，CMM/CMMIで書かれているプロセス領域の多くは，（元々が米国国防省のソフトウェア発注先の選定に用いられたという経緯もあり）受注型のソフトウェア開発組織で必要となるプロセス領域である．実際に 2010 年時点でも米国でCMMIのレベルを達成している企業のうち，半数以上が政府または軍事関連の受注企業である [51]．製品開発型のソフトウェア開発組織の場合，CMM/CMMIが示すプロセス領域だけでは不十分であるし，プラクティスでも過不足が多い[52]という問題がある．

ソフトウェア生産技術面で，CMM/CMMI の大きな特徴は，SEPG(Software Engineering Process Group)というソフトウェア開発組織でプロセス改善を推進する部署を定義していることである．

(b)各組織の目標に対応した改善フレームワーク

Basili の経験工場 [4]および GQM [53]は，組織の目標主導で定量化を推進する改善モデルである．経験工場とは，ソフトウェア開発の経験と成果物をその組織の資本として再利用することにより，改善を推進するようなソフトウェア開発組織である．品質改善パラダイムは，PDCA および SECI モデルからの影響がみられる．すなわち，特性記述，目標設定，プロセス選択，実行，分析，パッケージ化というサイクルを繰り返し，プロジェクトレベル，コーポーレートレベルの改善を推進する．

---

[3] 参考文献 [50]では，”Leveraging”となっているが，現在は”Learning”となっている．

[4] CMMI 序論[FM108.T102]では「組織内で使用される実際のプロセスは，適用分野，および組織の構造と規模など，数多くの要因に依存している．特に，CMMI モデルのプロセス領域は，典型的には，読者の組織で使用されるプロセスと 1 対 1 には対応しない.」と書かれている．

GQM（Goal, Question, Metric）は，組織においてソフトウェア開発に関わるメトリクスを制定するための方法論である．ソフトウェアの計測の前提には，組織としての目標があり，さらに目標を達成するための判断の基準を明確にする必要がある．このために，GQMでは，まず，概念レベル(Goal)で，測定を行う目的や測定対象（製品，プロセス，リソース）を定義し，運用レベル(Question)で，概念レベルで設定した目標を達成するために必要な質問を定義し，定量化レベル(Metrics)で，運用レベルで得られた質問に対して定量的に回答できるようなデータ群を定義する．Goal, Question, Metric の関係は以下の図 2- 3 のようになる．

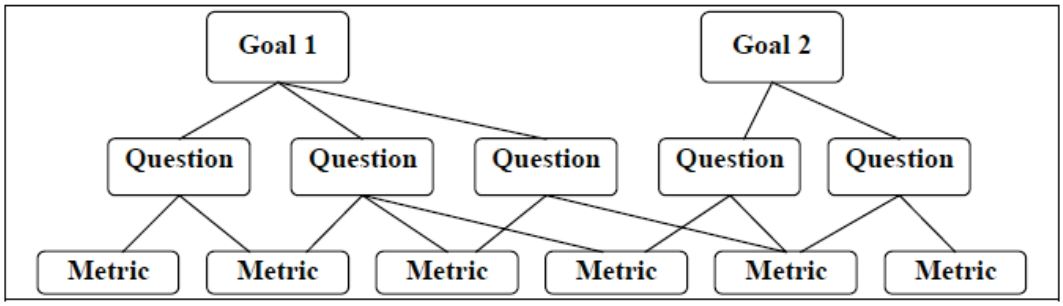

図 2- 3 GQM の構造 [53]

(c)定量化と改善を組み合わせたアプローチ

ソフトウェアプロダクトライン（SPL : Software Product Line) [54]は，共通的なベースとなるソフトウェア資産を形成し，それを戦略的に再利用することにより，多品種の製品開発の効率を抜本的に改善する手法である．複数製品で共通的に用いる「仕様(フィーチャ)」を部品化して品揃えし，再利用する．SPL はどの組織でも適用可能な方法論ではないが，その改善アプローチは，まず，(a)で述べたような基本的な構成管理や各種の定量化のしかけが前提となり，その定量化基盤を使って，(c）で述べたような組織の目標に従って資産化するソフトウェアを特定化し，継続的に資産を増やしていくという定量化と改善を組み合わせたアプローチを採用している．

2.3. ソフトウェア開発組織に関連する最近の話題

最近のソフトウェア開発組織関連の話題として，アジャイルソフトウェア開発手法および，バザール的なソフトウェア開発方法とソフトウェア開発組織の関係を述べる．また，ソフトウェア工学以外の話題として，情報管理，知識管理の課題を述べる．

(a)アジャイルソフトウェア開発手法

2000 年代に入り，アジャイルソフトウェア開発手法と呼ばれるソフトウェアを軽量かつ適応的にソフトウェアを開発する手法が数多く提案され，広く普及しつつある．序論でも

述べたとおり，これらの手法の多くは，経営工学での開発システム論や生産システム論の影響を受けており，組織的なソフトウェア開発方法と相反する考え方ではない．特にアジャイルソフトウェア開発手法の場合は，反復的な開発プロセス，要求の変更に迅速に対応するという特徴を備えているため，組織内でどのようにアジャイルソフトウェア開発手法を使ったプロジェクトを効率的に実施するかが，大きな課題になっている．この課題は，例えば「トヨタ生産方式に従ったラインをどのようにハードウェア生産システムの中で構築，変更するか」という課題に対応していると言ってよい．

(b)バザール方式によるソフトウェア開発

Raymond はエッセイ「伽藍とバザール」で，「バザール方式のソフトウェア開発」を提案した [55]．これまで大規模なソフトウェアは，中央集権的なアプローチによる開発（これを伽藍の建設に例えている)が必要だと考えられていたのに対して, Raymond はインターネットベースの開発コミュニティによるソフトウェア開発形態を分析し，不特定数のソフトウェア開発者が，バザールのような混沌の中でアイデアやソースコードをインターネットを介してコミュニティに持ち寄り，試行錯誤的にソフトウェアを開発していくバザール方式の開発方法が可能であることを示した．この開発方法が，今後あらゆるソフトウェア開発に適用が普及してきた場合，ソフトウェアの開発は本質的に「プロジェクトによる開発」になり，その上位構造として組織を仮定する必要が無くなる可能性がある．

しかし，以下の理由から今後もソフトウェア開発組織によるソフトウェア開発は無くならない．特定の目的に対応した要件で，決められたコスト，期限内に完成しなければならないソフトウェアの開発プロジェクトは今後も無くならない．そのようなソフトウェア開発プロジェクトで使用するバザール方式で作られたソフトウェアの条件は，すでに完成していることである．また，そのようなプロジェクトで新たに開発するソフトウェアをコミュニティベースでのソフトウェア開発方法を採用した例は調査しは範囲では見当たらない．すなわち，完成された部品としてバザール方式で作られたソフトウェアを部品的に使うことはありえても，ソフトウェア開発プロジェクトで今回開発しなければならない固有の目的のためにバザール方式は採用できないと考える．

このようなソフトウェア開発プロジェクトでバザール方式が採用されない理由は明白である．Linux をはじめ，オープンソースベースのコミュニティを使い，速く開発できた事例はある．また，高信頼性のソフトウェアがあるのも事実である．しかし，それは少数の事例であり，ある特定のプロジェクトを開発時に決められた納期以前に完成できることを保証するものでは決してない．また，高信頼，高品質を誇るバザール方式で開発されたソフトウェアであってもその開発スケジュールが商用ソフトウェア開発に求められる厳密さを求めることは困難である．

一方，ソフトウェア開発プロジェクトでの納期遅延は，プロジェクト失敗というだけでなく，そのソフトウェア開発組織の信用(credibility)を逸する結果となる．従って，商用ソ

フトウェアの開発にとって重要なのは，納期やコストの過去のベストプラクティスでは無く，これからの開発プロジェクトでリスクが発現する確率であるといえる．例えば，バザール方式のソフトウェア開発が仮に伽藍的な開発方法よりも平均 30%納期を早められたとしても，決められた納期を守れないリスクが他の開発方法よりも高ければ，Raymond のいうバザール的なソフトウェア開発方法を採用することは将来にわたって不可能である．

(c)ソフトウェア開発組織での情報管理，知識管理の動向

1978 年，米国でソフトウェアが著作権法の保護を受ける著作物となり [56]，その後，各ソフトウェア開発組織は，自社のソフトウェア資産が流出せず，組織外の不必要な情報がソフトウェアに混入しないような情報管理の仕組みを構築した．1990 年代までは，他社が知的財産権をもつソフトウェアは，特定の会社から導入したソフトウェアが大部分であり大きなリスクは無かった．しかし 2000 年代に入ると，ソフトウェア開発におけるオープンソースの利用が一般化し，組織内での知的財産権の管理が複雑化している．

また，米国におけるエンロン，ワールドコムなどの企業レベルの会計の意図的操作による不正行為に端を発し，ソフトウェア開発組織に限らず，企業の社会的な責任が重要視されるようになってきた．この一環として企業の中で会計監査制度の確立及び内部統制強化を求める動きが活発化した．米国のサーベンス・オクスリー法（SOX 法）に倣って，日本でも金融商品取引法が 2006 年に制定され，その一部として，IT 統制も求められている．

一方，企業が組織的かつ継続的に適用改善するモデルである Senge の「学習する組織」[57]や，企業がその組織内で知識を創成するためのフレームワーク野中，竹内の SECI モデル [49]は，ソフトウェア開発組織にも浸透してきている．特にソフトウェア開発においては，知識は高品質なソフトウェアを開発するのに最も重要な要因の一つである．従って，プロジェクトを跨った組織的な知識の共有による組織知の形成はソフトウェア開発組織の最重要課題のひとつである．このため，組織内でのコラボレーション基盤による個人のもつ暗黙知の獲得，伝達，組織での，共有や，形式知化を推進している．さらには，組織としての知識をソフトウェアという形にして，組織内で再利用できる知識とすることを目標としている．

効率だけを考えた場合，その組織の持つ複数のプロジェクト間で知識を共有し，再利用できる知識とすることが重要であるが，知的財産権保護，企業の社会的責任への対応という今日的な要求とどのように両立させるのかが現在のソフトウェア開発組織の課題になっている．この問題に対する本研究の取り組みは第五章で詳述する．

2.4. ソフトウェア開発組織での現状の課題まとめ

本節では，ソフトウェア開発組織に関連する経営工学，ソフトウェア工学の過去の研究を振り返り，現在のソフトウェア開発組織においてソフトウェア生産技術が過去のどのような工学や研究に関連し，影響されてきたかを概観した．ここまでの説明のまとめを次ページの図 2- 4 に示す．

大きな流れとして，基礎的な知識から，経営工学，ソフトウェア工学という流れがある．すなわち，統計学，システム工学，OR といった理学および基礎工学の知識を，企業組織という対象でどのように実践するかという観点で経営工学の知識が成立しており，ソフトウェア工学では，経営工学での知識をソフトウェア開発組織という対象に特化して知識化する傾向にある．ソフトウェア工学における組織定量化の研究の多くは，「生産工学に特化した経営工学」の影響を受けており，ソフトウェア工学における組織定量化の研究の多くは，「実践的システム工学としての経営工学」の影響を受けている．さらには，最近のソフトウェア工学のトピックであるアジャイルソフトウェア開発手法の多くは「部分最適から全体最適を目指す現在の経営工学」の影響を受けている．

このような流れにおいて，ソフトウェア開発組織での現状の課題を以下に示す．

（1） 「ソフトウェア開発組織の知識」という観点で経営工学とソフトウェア工学の従来の知識を統合体系化．特にソフトウェア開発組織の強みをプロジェクト横断かつ，継続的に発揮できるような組織的な仕掛けが必要．現状の組織的な取り組みの多くは各企業の暗黙知で，不特定の組織で活用可能なソフトウェア生産技術の統合体系化が必要．

（2） 多様化する環境（開発技術変化，開発プロセス多様化，法的な問題）に対応した組織的な開発の確立．アジャイルソフトウェア開発手法や，オープンソース活用といった多様化する開発プロセスへの対応とともに，組織の社会的責任や多様な知的財産権の問題と言ったコンプライアンス対応を両立したソフトウェア生産技術が必要

次節では，これらの課題のうち，(1)の「ソフトウェア開発組織の知識」という観点で経営工学とソフトウェア工学の従来の知識をどのように統合体系化できるかを述べる．

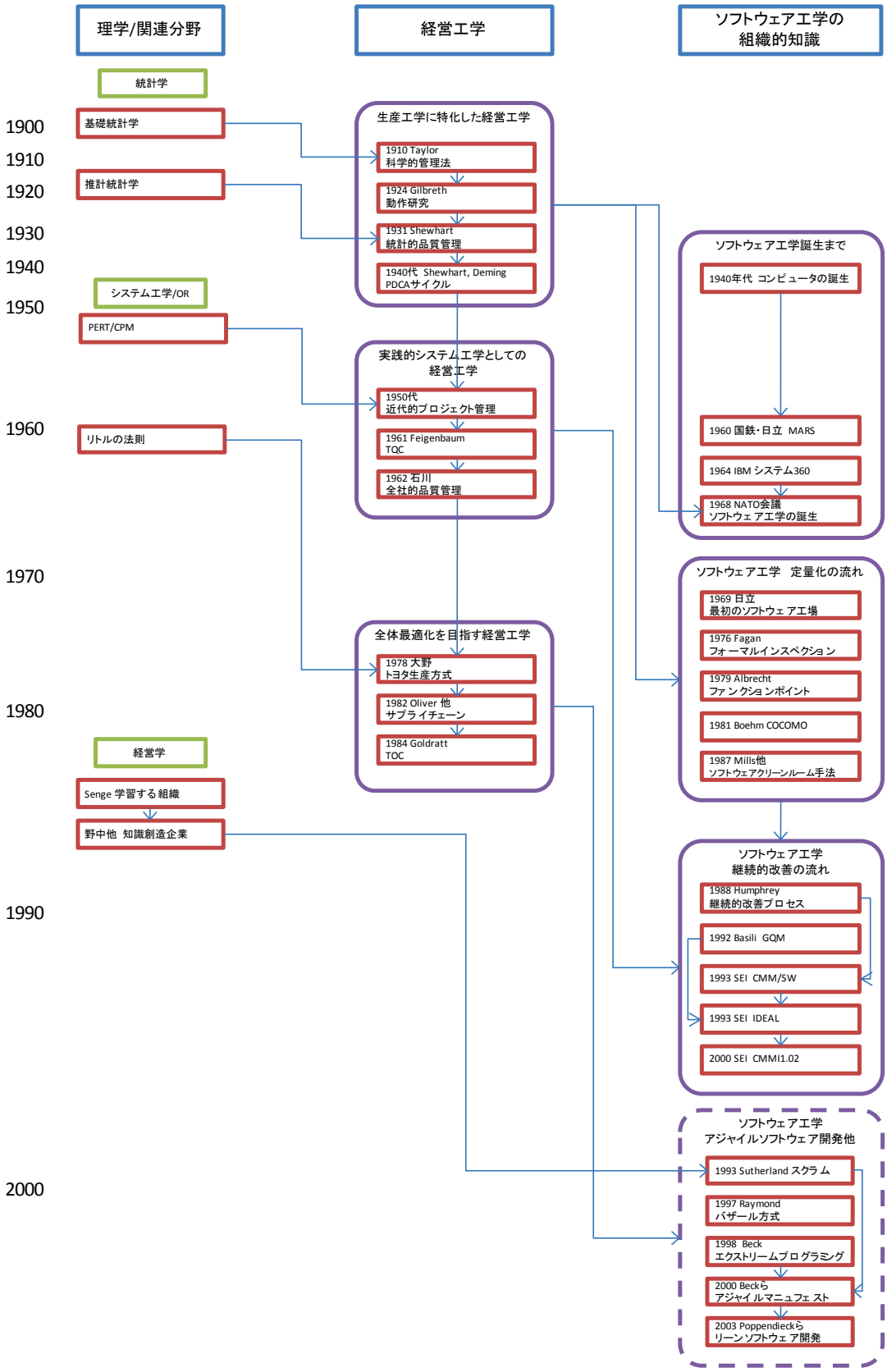

図 2- 4 ソフトウェア工学での組織の知識と関連する工学の流れ

3. ソフトウェア生産技術の体系

前節で述べたソフトウェア開発組織の課題に基づき，本節ではソフトウェア生産技術の統合体系化を行う．これまで暗黙知として各ソフトウェア開発組織の組織定量化や継続的な改善を統合体系として形式知化するにあたり，本研究では，前節で説明したように比較的形式知化されている「生産工学に特化した経営工学」，「実践的システム工学としての経営工学」の枠組みをソフトウェアにあてはめるアプローチで考えた．

すなわち，ハードウェアの生産技術でいう「生産の要素」や，改善，組織化といった業務機能をソフトウェア開発組織にあてはめ， 図 2- 5 のように体系化する．

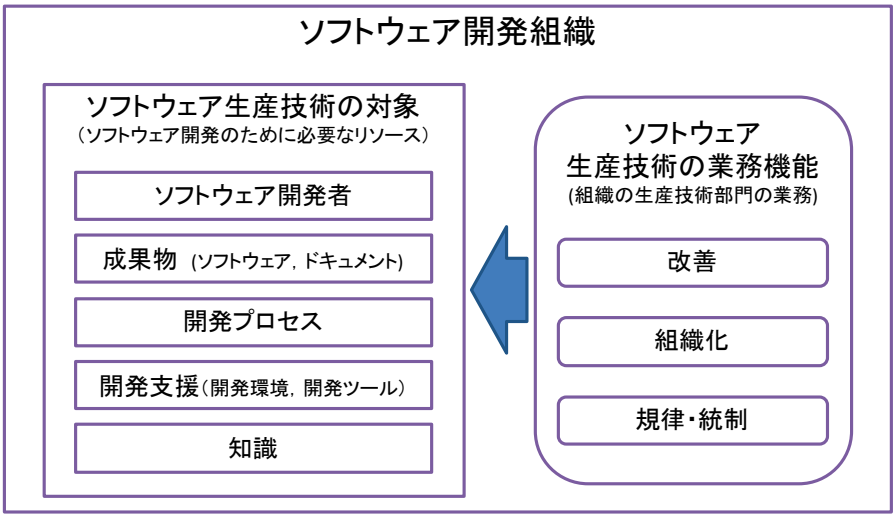

図 2- 5 ソフトウェア開発組織におけるソフトウェア生産技術の体系

ソフトウェア開発組織には，一つのソフトウェア生産技術を管掌する部門が存在し，改善，組織化，規律・統制という業務機能を持っている．業務機能の対象となるのは，ソフトウェア開発組織が持つソフトウェア開発に必要なリソースであり，生産技術の業務機能の対象として，組織レベルで改善，組織化，規律・統制が可能なものである．このような組織の持つリソースを，本研究では「ソフトウェア生産技術の対象」と名付ける．

4. ソフトウェア生産技術の対象と業務機能

前節で述べた体系に従い，ソフトウェア生産技術の対象と業務機能についての詳細を説明する． 表 2- 2 にソフトウェア生産技術の業務機能およびその対象ごとにどのような生産技術施策の例があるかをまとめた．

表 2- 2　ソフトウェア生産技術の業務機能と対象による施策の例

対象 / 業務機能	ソフトウェア開発者	成果物（ソフトウェア，ドキュメント）	開発支援（開発環境，開発支援ツール）	プロセス	知識
改善	技術教育	機能向上，品質向上	生産性/品質が向上するツール使用 オフィス環境改善 マシン環境改善	プロセス改善 期間短縮	知識創造(SECI) 知識交流の場の改善
組織化	スキル認定	部品再利用 コーディング基準 メトリクス	標準ツール 共通ツール 環境の共用	開発規格 標準プロセス 作業手順書 フォーマット	設計ガイド 共通の知識交流の場の設定
規律・統制	技術教育 倫理教育	契約違反のソース混入防止	開発エリアの分離等 危険なツール使用制限	開発規格	契約に基づく分離

本節では，ソフトウェア生産技術の対象を説明し，その後にソフトウェア生産技術の業務機能ごとにどのような施策が必要かを述べる．

4.1.　ソフトウェア生産技術の対象

経営工学においては，生産に必要な要素を，3M(Man, Material, Machine)または 4M(3M に加えて Method)と呼ぶことが多い．ここでの 3M，4M は，品質管理における石川の特性要因図の説明 [25]に基づく．ここで，「要素」という用語はあいまいなので，本研究では，ソフトウェア生産技術の業務機能を実施する対象（以下，「ソフトウェア生産技術の対象」）と呼ぶ．また，ソフトウェア開発においては，ハードウェア生産との差異を考慮し，本研究では，ソフトウェア生産技術の対象を表 2- 3 の網掛け部分の五つとする．

表 2- 3　ハードウェアとソフトウェアの生産技術の対象

ハードウェア生産技術の４M	ソフトウェア生産技術の対象
人(Man)	ソフトウェア開発者
部品(Material)	成果物(ソフトウェア，ドキュメント)
生産機械(Machine)	開発支援（開発環境，開発支援ツール）
方法(Method)	開発プロセス
―	知識

ハードウェアの４M との大きな差異は，ハードウェアにおける「部品」を，ソフトウェアでは「成果物」としている点，ハードウェア側には無い「知識」をソフトウェア生産技術の対象にしている点である．以下，ソフトウェア生産技術の対象について説明する．

（1）ソフトウェア開発者

ソフトウェア開発プロジェクトの成果物（ソフトウェア，マニュアル）および，中間成果物（設計ドキュメント等）に直接関与（設計，コーディング，テスト，ドキュメント作成，製品検査等）する技術者をいう．Boehm は COCOMO [40] [58]の提案において，ソフトウェア開発者の能力が，プロジェクトのコスト，納期等に最も多く影響を与えていることを示した．また，ソフトウェア技術者の個人差によって，ソフトウェアの生産性，品質が大きく異なることも知られている [15].

1960 年代から 1970 年代にかけての日本のソフトウェア工場では，ソフトウェア開発者をハードウェア工場の製造に携わる労働者に見立て，均一なスキルを前提にしたソフトウェア開発プロセスの構築を目指した [35] [59]．また，1980 年代にはCOCOMOによって，ソフトウェア開発者の能力係数を導入し，ソフトウェア開発者の能力差を正規化することにより，プロジェクトのコストや納期を見積もる手法を提案した [40]．現在では，これらの手法とともに，ソフトウェア開発者のスキルの差は認めたうえで，いかに適材適所に人材を割り当て，また優秀な人材ができるだけ多くの成果物に関与できるような生産システムを構築するかが生産技術上のキーポイントとなっている [5].

（2）成果物（ソフトウェア，ドキュメント）

ソフトウェア開発プロジェクト全体の成果物，または，その中の各工程の中間的な成果物を含む．具体的には表 2- 4 のものである．

表 2- 4 成果物の分類

成果物	説明
プログラム	ソースプログラム, オブジェクト, ロードモジュール等. テスト用のプログラムも含む
ドキュメント	開発用ドキュメント，保守用ドキュメント，取扱説明書（マニュアル）等
その他	開発記録，テストデータ等

ハードウェア製品では生産の要素として部品が極めて重要である．なぜなら，部品の組み合わせによって最終製品ができるので，部品の存在が必要不可欠だからである．一方，ソフトウェア製品では必ずしも部品は必要不可欠ではない．しかし，ほとんどソフトウェア開発プロジェクトでは，そのプロジェクトの入力として過去のバージョンのソフトウェア自身が前提条件になっている．また，ソフトウェア開発の各工程のドキュメント等の中

[5]本研究では，有能な人材の採用や採用した人材の育成はソフトウェア生産技術の範疇としない．多くの企業では生産技術部門が採用や教育部門から独立していることによる．

間成果物が次の工程の入力となっている．このような，ソフトウェアの特徴を踏まえ，ソフトウェア生産技術ではハードウェア生産での部品に相当するものを，「成果物」とする．

成果物としての，各工程の中間成果物とソフトウェア開発プロジェクト全体の成果物を説明する．まず，中間成果物は，次の工程の入力になるため，ハードウェア生産における部品または半製品の位置づけに相当する．さらに，ソフトウェア開発プロジェクト全体の成果物も，ソフトウェアのライフサイクルを考えた場合，そのソフトウェアに対する機能拡張や移植をする別プロジェクトの入力となるため，ソフトウェア生産技術の対象とする．

ソフトウェア生産技術のビューで良い成果物とは，第一にステークホルダにとって価値があること，すなわち高品質であることである．ここで，ステークホルダとは，そのソフトウェアの利用者だけでなく，開発者や，将来的にそのソフトウェアを保守する技術者も含む. 従って, そのソフトウェアの利用時の品質特性や製品の外部/内部品質特性といった, ISO-9126 [60]に記述された品質特性を満足することが必要である．

良い成果物の第二は，再利用可能なことである．ソフトウェア生産システムの入力として使うことができるソフトウェアを充実させることにより，将来のソフトウェア開発プロジェクトで新規に開発するソフトウェアの量，すなわちコストを削減できる．この目的を達成するために，ソフトウェア開発組織全体で，再利用の仕組みを作り上げることもキーポイントとなる．

（３）開発支援（開発環境，開発支援ツール）

ハードウェア製造の場合，生産の主力は機械であるが，ソフトウェア開発の場合は，いまだに人間（ソフトウェア技術者）が主力である．従って，ソフトウェアの生産技術での開発環境や開発支援ツールの位置づけは，人間の効率を上げることである．表 2- 5 にソフトウェア生産技術の対象となる主な開発環境および，開発支援ツールの分類を示す．

表 2- 5　開発支援の分類

大分類	中分類	主なソフトウェア生産技術の対象
開発環境	個人環境	作業環境
	計算機環境	デスクトップ，サーバ，開発クラウド等
	コラボレーション基盤	会議室環境，電子会議室 情報共有の場（物理的な場，電子的な場）
開発支援ツール	開発ツール	プログラム生成ツール
		設計支援ツール
		テスト支援ツール
	管理ツール	プロジェクト管理，成果物管理，構成管理，品質管理ツール

ソフトウェア生産技術のビューで，良い開発環境，開発支援ツールとは，開発者がソフトウェア開発作業に専念できること ，結果として良い品質の成果物が開発可能なことである．さらに，単に，良い開発環境，開発支援ツールということだけではなく，開発プロジェクトをまたがった組織全体で使える開発支援ツールや開発環境を構築してコストを削減したり，組織全体で開発環境を共通化することにより人の流動性を良くしたりするという点もキーポイントとなる．

（4）開発プロセス

ソフトウェアを開発するために必要な工程の組合せの構造をソフトウェア開発プロセスという [1]．小規模のソフトウェアの場合，一つの工程でソフトウェアが完成する場合もあるが，コンパイル単位が多数ある中規模以上のソフトウェアの場合，機能設計，詳細設計，コーディングといった複数の工程を設定する，さらに複数コンポーネントをどのような順番で開発するかの設定が必要で，それらの工程をどのように組み合わせるかが課題となる．

ハードウェアの場合，一つの生産プロセスの上に通常多数の部品，製品が流れるが，ソフトウェアの場合は，一つ一つのソフトウェア開発プロジェクトごとに開発プロセスは異なる．しかし，プロジェクトを初期化するたびにプロセスをすべて再設計するということはない．ソフトウェア工学では，工程の組合せの構造をパターン化したソフトウェア開発プロセスモデルが提案されている．また，一つ一つ異なるソフトウェア開発プロセスをどのように導き出すかというソフトウェア開発プロセス導出モデルも提案されている．主なプロセスモデルおよびプロセス導出モデルを表 2- 6 に分類した．なお本研究では，自組織でのソフトウェア生産技術を適用範囲とするため，表 2- 6 の組織外開発部分については適用範囲外とする．

多くのソフトウェア開発組織では，開発規格，または開発ガイドというドキュメントに組織内で開発する標準的なソフトウェア開発プロセスが定義されている．過去，日本のソフトウェア工場では，組織内の規格として開発プロセスを厳格に決め，各ソフトウェア開発プロジェクトはそれに従って工程を設計した．しかし，現在では，日本の主要企業でも，変更可能なガイドとして指定しておいて，プロジェクト毎にカスタマイズ可能にする場合が増えている．この場合，特定のソフトウェア開発プロジェクトは，これらのガイドを参考に開発プロセスを設計する．

ソフトウェア生産技術のビュー，すなわちプロジェクトを多数実行させるようなソフトウェア開発組織にとって，良い開発プロセスとは，まず，所定の QCD が達成できるプロセスであること，次に，管理可能なこと（プロジェクト毎にばらつきが少ないこと，コスト，スケジュール，生産性，品質の見積もり可能なこと，評価が可能なこと）が挙げられる．ソフトウェア開発の場合，一つ一つのプロジェクトは違った特性を持つため，本質的に複数ソフトウェア開発プロジェクトのあらゆる結果には，ばらつきが生じる．しかし，それらの結果を組織的に管理可能にするために，プロセスの中で標準化できるものは標準化し

て，スケジュール，生産性，品質などが，組織として見積もり可能にすることが重要になる（ばらつきを管理可能にするためのもう一つの施策，正規化は組織化の業務機能のほうで説明する）．

表 2- 6 ソフトウェア開発プロセスモデルの分類

大分類	中分類	モデル名	説明
プロセスモデル	逐次形	ウォーターフォールモデル	一連の工程を順序的に実行するモデル．一つ一つの工程を確実に実行してから次の工程を行う．
	反復型	インクリメンタルモデル	基本設計とコア部分の実装を行った後，段階的に小機能を確認しながら追加していく．
		エボリューショナリーモデル	試行錯誤的に実装，評価を反復実行して，より良い機能を実装していく
	統合型	RUP [61]	反復的に行う実装主体のマイクロプロセスと開発プロセス全体を大域的な観点で逐次実行するマクロプロセスで統合
	組織外で開発	COTS(commercial off-the-shelf)	商用パッケージ製品の購入
		発注モデル	他組織に発注してソフトウェア開発
		バザール方式 [55]	インターネット上のコミュニティでの開発者を限定しない開発
プロセス導出モデル	規格による統一	日本のソフトウェア工場 [3]	組織全体で詳細に決められたプロセスのパターンで実行
	リスク主導	スパイラルモデル [62]	そのプロジェクトの一番大きなリスクを解消する工程から実行するようにプロセスを設計
	スループット主導	TOC のクリティカルチェーン [63]	そのプロジェクトのスループットを最大にするようにプロセスを設計
	OR 手法	CPM/PERT [22]	そのプロジェクトのリソースと必要な工程と依存関係から工期を推定．これを調整することによりプロセスを設計

（５）知識

ソフトウェア生産技術の対象の最後は知識である．ソフトウェア開発組織での知識の分類を表 2- 7 に示す．

表 2- 7 ソフトウェア開発組織における知識例

大分類	説明
暗黙知	ソフトウェア開発技術者個人が持つノウハウ，開発スキル
	ソフトウェア開発プロジェクト管理者が持つノウハウ，管理スキル
	各種支援部署（ソフトウェア生産技術も含む）のノウハウ，スキル
	組織が(明文化されていないが)その文化として持っているノウハウ，スキル
形式知	組織全体のプロセス規格，コーディングガイド，設計ガイド等
	部署内の規格，プロジェクト規約，作業手順書等

ソフトウェア開発に必要な知識の多くはソフトウェア開発者がそれぞれに持つ暗黙知である．これは，ソフトウェア開発における生産性の最も大きな差異要因はソフトウェア開発者のスキルであることからも明白である．このソフトウェア開発者が持つ暗黙的なソフトウェア開発知識をどのように，プロジェクト，組織と言う単位に広め，それをどのように組織としての形式知化し，それを，各プロジェクト，各ソフトウェア開発者に広めていくかと言うのはソフトウェア生産技術の最も重要な業務機能の一つである．

一方，ソフトウェア開発に関連する暗黙知の多くは形式知化できないし，形式知化できる知識でも，日進月歩のソフトウェア技術の進化により，形式知化するころには陳腐になっている場合が多い．従って，ソフトウェア生産技術のビューで「良い知識」を考えた場合，他の対象のように定義されて管理できることのみを重視することはできない．すなわち，ソフトウェア生産技術における知識は，定義可能，管理可能であることも重要であるが，実利優先で，暗黙知であっても広く適用できることが必要である．また，形式化するとしても，寿命の比較的長い規格ではなく，設計ガイドといった方法をとるのが現実的である．

ソフトウェア開発においては，知識そのものの話とともに，組織の中でどのように知識を蓄積し，どのように知識を共有し，どのように知識を創成するかという，知識の場が重要である．ソフトウェア開発における特徴的な知識の場は，知識の交流を促進するとともに，知的財産権の保護といった，各種規制の問題をクリアできる必要がある．この課題については，五章で詳細に述べる．

以下に，ソフトウェア生産技術のビューでの良い知識をまとめる．

- 結果として良い成果物が開発できる知識
- 暗黙知または形式知で組織に広げることが可能な知識

・ 個人が簡単に修得可能で，すぐに役にたつような知識
・ 上記を実現でき，かつ知的財産権の問題をクリアできるような組織内の知識の場

これらの知識及び知識の場をソフトウェア開発組織に蓄積，構築することにより，他のソフトウェア生産技術の対象の価値も上がるという副次的な効果もある．

4.2. ソフトウェア生産技術の業務機能

本研究ではソフトウェア開発組織が持つソフトウェア生産技術活動を，大きく以下の３つの業務機能「ソフトウェア生産技術の改善」「ソフトウェア生産技術の組織化（共同化，定量化，標準化，共通化)」,「ソフトウェア生産技術の規律・統制」に分類する．本項では，３つの業務機能がどのようなものであるかを説明する．ソフトウェア生産技術の業務機能の具体的な事例は，三章で述べる．

（１）ソフトウェア生産技術の改善

組織の持つソフトウェア生産技術に対する継続的な改善能力である．従来のソフトウェア工学では，一つ一つのソフトウェア開発プロジェクトの QCD を向上させるプロジェクト管理に焦点を当てていた．これに対して，本研究では，組織的開発におけるソフトウェア生産技術という観点で，組織の持つソフトウェア開発プロセスを改善し，その上に流れる不特定多数のソフトウェア開発プロジェクト全体での効果に焦点を合わせる．本項では，どのように，組織的に技術改善のサイクルを回すかという観点で，従来の研究の概略を述べ，本研究で提案するソフトウェア生産技術による継続的な組織改善のモデルを説明する．

（ａ）従来の研究での改善モデル

ソフトウェア工学の過去の取り組みでは，問題定義が先か，定量化が先かの考え方により大きく二つの改善モデルがある．

定量化前提の改善モデル

定量化前提の改善モデルとは，定量的に現状を明確に把握できる手段をまず組織に構築し，そこで得られたデータから，組織自体やプロセスの問題点を明確にして改善していくモデルである．この場合，改善の前提条件として定量化がある． CMM/CMMI，エンピリカルソフトウェア工学 [64]がこのタイプの改善モデルである．CMM/CMMI は，どの組織でも当てはまるような基本的な定量化を前提としている．一方，エンピリカルソフトウェア工学は，ある特定の組織の問題点を抽出するようなアドホックな定量化を前提としているという違いがある．

このモデルは，モデル適用時点でソフトウェア生産技術的に成熟していない組織または，受注型のソフトウェア開発ビジネスを行っている組織に有効である．ソフトウェア

生産技術的に成熟していない組織の場合，まずは，ソフトウェア開発に関わる基本的なデータを採取することによって，その組織のパフォーマンスや製品品質などの問題点が分かる場合が多い．従って，そのような組織では，まず共通的な指標を採取する仕掛けを構築することが前提条件になる．また，ソフトウェア受注型のビジネスの場合，製品開発型のビジネスと比較して業界の中で求められるソフトウェア開発プロセスのバリエーションが大きくなく，CMMI で求める定量化の仕掛けが組織の目標に合致する場合が多い. また, CMMI が受注の前提条件の場合, CMMI の適用は手段ではなく目標となり，まず，CMMI で求めている定量化の仕掛けを構築することが先決となる．

問題定義前提の改善モデル

問題定義前提の改善モデルとは，組織の目標，問題認識をまず定義し，それに従って，どのようなデータを採取し，どのようなメトリクスを使い，どのような評価指標を確立するかを決めて改善していくモデルである．この場合，定量化は組織固有の目標を達成するための手段となる．2 節で述べた過去の研究 GQM，TOC，QC(PDCA)がこのタイプの改善モデルである．

このモデルは，汎用的なソフトウェア製品開発組織や，組み込みシステムを開発する組織に有効である．CMM/CMMI では，すべてのプロセス領域でレベル４（管理可能なレベル）までプロセスを確立した後に，レベル５（最適化しつつあるレベル）がある（表 2- 1 参照）．しかし，実際には，CMM の全てのプロセス領域でレベル５にしなければ継続的な改善ができないわけではない．さらには，製品開発組織の必要なプロセスを考えた場合, CMM のレベル５で記述されている具体的なプラクティス以上のプロセス成熟度をもつプラクティスが必要な場合もある．

（ｂ）従来の研究での課題

定量化前提の改善モデル，問題定義前提の改善モデルはそれぞれ問題がある．定量化前提の改善モデルは，開発プロセス成熟度の低いソフトウェア開発組織には有効なモデルであるが，例えば，CMMI のレベル３(定量化されたレベル)の求めている定量化を達成した場合でも，レベル５でいう継続的な最適化を可能にするためには，CMMI が求めている定量化では不足である [52]．この問題の本質は，CMMI などの定形的なモデルが，そのモデルを使う複数のソフトウェア開発組織の事業の方針にそれぞれ違いがあることを過小評価していることにある．このため，CMMI のレベル４，５となるとあるプロセス領域では事業の求める成熟度からみて不足していたり，あるプロセス領域では過剰になったりする．

問題定義前提の改善モデルは, 開発プロセス成熟度の高い組織には有効である．しかし，成熟度の低い組織においては，基本的な定量化の仕掛けができていないため，問題定義前提のアプローチを採用すると，問題定義からメトリクス制定の部分（GQM の QM の部分）で行き詰まる，また，目標毎に基本的な定量対象，例えば，成果物の量や品質の測定単位

が，各々の目標ごとに違う採取方法や違うメジャーになってしまい，測定負荷が増えたり，組織としての集計が困難になったりするという危険性もある．問題定義前提の改善モデルの本質的な課題は，一つ一つの事業目標を絶対視していることで，実際の組織運用を行う立場では，個々の目標から独立な定量化も必要であるし，複数の目標を効率的に計測できるような定量化も必要である．

また，両方の改善モデルとも，組織の中での推進方法は，トップダウンに展開することを暗黙的に仮定している．実際のソフトウェア開発組織では，CMMI のような定形的な処方箋や，GQM のような組織目標からの展開だけではなく，実際にソフトウェアを開発している開発者側のビューで，ボトムアップに組織的な課題が見つかる場合もある．

（ｃ）本研究で提案する継続的改善モデル

本研究では，従来研究の課題に対応するため，継続的な改善モデルとして，以下の 2 点の基本方針を設けた．

定量化指向と問題定義指向を組み合わせたモデル

定量化指向の改善モデルと問題定義指向の改善モデルそれぞれの長所を取り入れた改善モデルを提案する．ソフトウェアプロダクトラインでは，まず定量的な基盤（CMM2~3 の成熟度）を構築し，その後は，問題を定義し，それに対して改善していくモデルを提案している．ただし，ソフトウェアプロダクトラインでの提案は組織の再利用可能なソフトウェア資産の蓄積（本研究の用語では「成果物」）という特定目的のための方法論であるのに対して，本研究ではソフトウェア生産技術の他の対象にも適用可能な改善モデルを示す．

トップダウンとボトムアップ両面から問題定義

ソフトウェア生産技術面での組織の問題定義は，基本的には，組織全体の目標と現状のギャップを知ることにある．組織の目標が明確に定義されていれば，その目標からトップダウンに改善する．一方，組織全体の課題であっても，想定していないような課題を摘出するようなことも多い．そのような場合は，エンピリカルソフトウェア工学の考え方を適用し，基本的な定量化の組合せにより，その組織のプロセスの問題点を抽出し，それを起点にボトムアップに改善するパスを設ける．

以下，本研究で提案する継続的な改善モデルのステップを述べる．

STEP1　組織レベルの基本的な定量化

このステップでは，ソフトウェア開発組織が QCD のレベルの基本的なデータ項目を採取できるようにして，組織生産性，品質密度等の基本的なメトリクスの測定を可能にする．採取すべき基本項目は下記である．

- 開発したソフトウェアの成果量
- ソフトウェア開発にかかったコスト
- 開発したソフトウェアで摘出されたソフトウェアフォールト
  - 開発中のソフトウェアフォールト
  - 開発後のソフトウェアフォールト

開発プロジェクト毎の基本的なデータを採取可能にするだけでなく，それを組織全体の数値として集計可能になっている必要がある．また，採取時には，計測負荷ができるだけ低いような仕掛けが必要である．各プロジェクトでどのようにデータを採取するか，各プロジェクトで採取したデータをどのように組織全体として統合するかという事例は三章で述べる．

STEP2　改善目標の設定

このステップではソフトウェア開発組織の中のさまざまな活動から改善が必要なものを抽出する．ソフトウェア開発以外のプロセスからの意見，従業員が随時問題点を指摘できる仕掛け作成も重要である．重要度，緊急度，コスト対効果から改善項目を選定する．また，改革項目相互に依存関係がある場合は，それに従う．改善目標を得るために考慮すべき項目を表 2- 8 に示す．一般的に，その組織固有の事業方針や現状の組織的な問題点に関わる項目の優先度が高い．したがって，この表の上の項目ほど優先度が高い．

表 2- 8 改善目標設定の考慮すべき項目

#	考慮すべき項目	説明
1	事業方針	組織の事業方針．製品企画プロセスからの入力.
2	市場からのクレーム，障害解析	組織のフィールドでの品質保証プロセスから入力．ソフトウェア製品の不具合に起因する障害だけでなく，市場からのクレーム等も.
3	プロセス監査結果	組織のプロセス監査結果．ISO-9001，CMMI，日本経営品質賞等に従った自組織のプロセスの監査結果.
4	他社ベンチマーク結果	市場でのベストプラクティスとの比較
5	従業員の意見	ボトムアップの意見収集結果
6	CMMI ギャップアセスメント結果	CMMI で決められた定型的なプロセス領域での問題点抽出

STEP3　改善目標を計測するためのメトリクス作成(GQM)

このステップでは，STEP２で立てた改善目標から GQM [53]の方法論に従い，組織内で

継続的に定量管理できるようなメトリクスを導出する．組織単位で使用可能な定量化の要件については，ソフトウェア生産技術の機能要件の一つである「組織化」の項で述べる．

多くのメトリクスは，STEP1 の基本項目の組合せから導出可能である．その場合は，計測負荷は大きく変わらない．STEP1 の基本項目だけでは，改善目標を定量化するのに不足または不適切な場合，基本項目の採取条件，計測方法等を変えたり，新たな項目を採取したりする． 基本項目を変える場合，（計測負荷が変わらないことを条件に）変更前のメトリクスと変更後のメトリクスが相互変換可能にして，連続性を保つ．すなわち，従来のメトリクスでも計測可能なようにしておく．さらに，できれば新メトリクスで，変更前の値が計測/推測できるようにすることが重要である．

STEP4　設定したメトリクスを使って継続的改善（PDCA）

このステップでは，各プロジェクトで設定したメトリクスを採取し，そのプロジェクト，部署，組織全体を評価する．プロジェクトの単位，組織の単位で PDCA サイクルを回す．プロジェクトの単位での PDCA サイクルは必然的にプロジェクトの開始終了時期に依存する．一方，部署や組織全体の PDCA のサイクルは独自に設定可能である．一般的に，組織全体の経営のサイクルに合わせるのが望ましい．例えば 4 半期ごとに事業経営のチェックを行うような組織では，それに合わせてソフトウェア生産技術の PDCA もまわし，Plan の部分では，その時点での事業方針に従って，ソフトウェア生産技術の対象となるソフトウェア開発者, 成果物(ソフトウェア，ドキュメント), 開発支援 (開発環境, 開発支援ツール)，開発プロセス，知識のそれぞれについて計画を立て，期間中 Do で実行後，期間末に Check-Act として事業的な課題，対策と，ソフトウェア生産技術的な課題，対策がマッチしているのか確認するのが良い．

STEP5　定期的なメトリクスのスクリーニング

このステップでは，定期的に，その時点での事業方針と採取しているメトリクスが対応しているかどうかをチェックし，改善または不必要なものは廃止する．有期限のメトリクスの例としては，移行期間を設けたような新施策の適用率の管理がある．新施策の適用プロジェクトを漸増させるようなフェーズでは，適用推進のためのメトリクスが必要であるが，ある期間を経れば新施策を使用する条件や条件に従った使用の強制ができるようになり，そのメトリクスは廃止できる．スクリーニングのタイミングも，事業経営のサイクルに合わせて実施するのが良い．

STEP２～STEP5 を繰り返す

STEP2 から STEP5 までを，事業目標設定，実行，評価といった，事業経営のサイクルに合わせて繰り返す．

本研究で提案するソフトウェア生産技術の継続的な改善モデル全体のサイクルを図 2- 6 に示す.

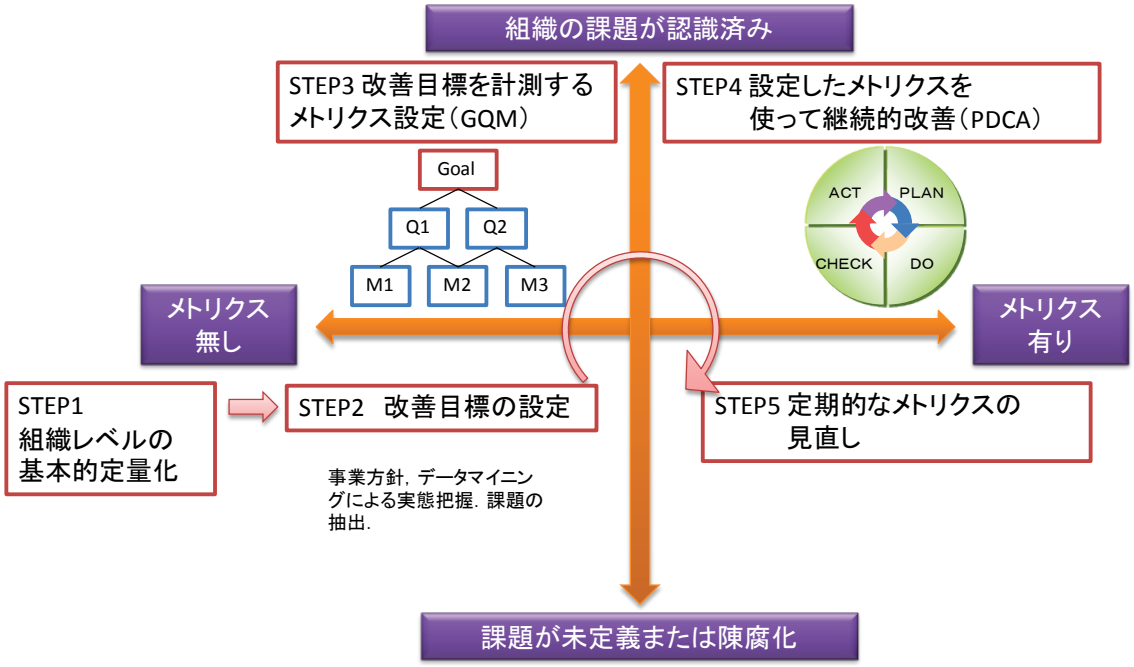

図 2- 6 本研究で提案するソフトウェア開発組織の改善モデル

（２）ソフトウェア生産技術の組織化

二つ目の業務機能は，ソフトウェア生産技術の組織化である．この業務機能はソフトウェア生産技術を組織に定着させ，組織全体としての強みにする活動である．本項では，組織化の目的及び，組織化の分類を述べる．

（ａ）組織化の目的

全体としての効率化

組織化の第一の目的は，個人やプロジェクトレベルの部分最適ではなく組織全体としての成果や効率を最大化することである．すなわち，組織全体で，業務や，成果物，知識を集約することにより，組織全体の QCD 向上，ソフトウェアの機能向上および不具合の最小化，顧客満足の最大化，コストおよび労力の最小化，納期の短縮を図ることである． 例えば，各プロジェクトでばらばらに持っていた構成管理のタスクを組織全体で共有したり，各プログラムでばらばらに開発していた共通機能を部品として再利用したりすることにより，一部のプロジェクトでは非効率になったとしても，組織全体の効率を向上し，コストを削減することを組織化は目標とする．

組織レベルで管理可能

組織化の第二の目的は，個人やプロジェクトレベルではなく，組織レベルで管理可能にすること，または，組織としてのリスクを最小化することである．このためには，単に定量化できるだけでは無く，組織全体で評価が可能であることが必要である．さらには，組織として管理可能にするためには以下のような要件を満たす必要がある．

- 平均値，メディアン，モードを向上するだけでなく，ばらつきを最小限にする（ばらつきの多いメトリクスは，定量化の効果が少ない）
- 複数プロジェクトでの測定結果が比較でき，かつ複数のプロジェクトの結果をマージして組織の結果として測定できること
- データの採取やデータの集計など，管理コストが最小限とすること

多くの場合，組織全体を管理可能にするために，ソフトウェア開発プロセスや，ソフトウェア開発環境を変えたり，各プロジェクトの集計時にばらつきを考慮した正規化を図ったりすることも必要である．この事例は 3 章で詳述する．

（b）組織化の分類

組織化の分類を表 2- 9 に示す．一般に，この表で上の項目ほど，初期段階から導入可能な組織化で，下の項目ほど，より高度な組織化である．

表 2- 9 組織化の分類

分類	説明
共同化	プラクティス共有レベル．暗黙知としてのノウハウをプロジェクト横断的に組織内で共有
定量化	組織として管理可能な定量的なメトリクスで管理
標準化	ガイド，規格といった組織内の形式知として組織内普及，徹底
共通化	知識共有からモノ共有（標準化された別々のシステムを使うのではなく，同じシステムを組織全体で共有する）

以下に，組織化の分類の詳細を説明する．

共同化

共同化とは，特定のソフトウェア開発プロジェクト内でインフォーマルにプラクティスが共有されている状態から，プロジェクト横断的に組織内でそのプラクティスを共通化することである．多くの場合，これらのプラクティスは暗黙知で，明文化されていない．共同化により，プロジェクトそれぞれの QCD 向上や，組織の定性的な評価は可能であるが，

プロジェクトおよび組織全体の定量的な評価はできない.

定量化

定量化とは，各ソフトウェア開発プロジェクトで採取する開発時のデータや，そのデータを加工して得られるメトリクスを各プロジェクトで共通にすることにより，複数のプロジェクトでの比較を可能にすることである．具体的には，ソフトウェア開発に関わる QCD や QCD を計測するための基本的なデータ，すなわち，生産量，コスト，時間，開発期間を計測可能にする．表 2- 10 にソフトウェア生産技術の対象毎の定量化の例を示す.

表 2- 10　ソフトウェア生産技術で必要な定量化例

対象	定量化の例
ソフトウェア開発者	標準単価（¥/h）
	能力係数
成果物	ファンクションポイント
	ユースケースポイント
	LOC（ソース行数）
開発プロセス	工程別の基準値
	工程別のソフトウェアフォールト数

プロジェクトおよび組織の観点で，ソフトウェア生産技術の対象を評価可能にするためには，単に定量化できるだけでは不十分である．定量化の目的は，定量的に問題把握が可能で，定量的に組織の制御が可能で，かつ定量的に組織達成したかどうか判断可能なことである．また，組織的に集計したメトリクスを使い，月，年といった単位で，その組織がどれだけ改善ができているのかが計測できるようにすることが必要である．したがって，組織全体で使用できる定量化は以下の条件を満たす必要がある.

- 事業方針に照らして意味があること（例えばコスト削減，売上向上等）
- 一つ一つのプロジェクトの結果が，他のプロジェクト結果と比較可能で，かつ，組織として，複数プロジェクトの結果がまとめられ，継続的に改善していることを定量的に評価できること.
- プロジェクトの性質の違いを考慮して，採取したデータを正規化したうえで集計・評価ができるようになっていること
- 結果指標だけでなく，推進指標として設けられること
- 現場での実感に合っていること
- 定量化で使用する測定項目やメトリクスが不必要になったときに，捨てることができること．このために，存続の条件があること.

標準化

標準化とは，ソフトウェア開発組織において組織内のプロジェクトで成果物の形式，ソフトウェアの開発方法，ソフトウェアの開発環境などを同様にすることである．標準化することにより，組織全体の開発環境に対するコストを減らし，組織内でのばらつきを減らすことによって見積もりが容易になる．また，組織全体のリソースの流動性を高まるのも標準化の特長の一つである．

例えば，組織全体で設計ガイド，開発規則を設け，プロジェクトは異なっていても，同じような開発方式，設計を行うのが標準化である．また，同種のソフトウェアテストツールが市場に複数出回っていて，その効果に大きな差が無い場合，その中の一つのツールをその組織の標準テストツールにして，組織内の各プロジェクトに使用させる．そうすることによって，導入，運用コストの削減，そのテストツールのベターユース等の使用ノウハウの蓄積を図ることができる．また，そのテストツールを習熟したソフトウェア開発者が他のプロジェクトでそのスキルを生かすことが可能になるという組織内の人材流通のメリットもある．

共通化

共通化とは，ソフトウェア生産技術の対象を「もの」レベルで同じものにすることである．前項で述べた標準化は，各プロジェクトで同じ（ような）ものを使うがインスタンスとしては違う組織化であった．これに対して，共通化はインスタンスのレベルで組織内共通にする．例えば，設計ガイドで同じように設計するのが標準化である．これに対して部品レベルで組織内共通に使用するのが共通化である．その他に，組織内のソフトウェア開発プロジェクトで，共通的なプロセスを設けたり，共通的なソフトウェア開発環境を複数プロジェクトで共有したり，設計者が共通にアクセス可能な知識ベースを提供したりすることも共通化である．組織内のプロセス，成果物を共通化することによって，組織全体のコストが抜本的に削減される．

（３）ソフトウェア生産技術の規律・統制

三つ目の業務機能は，ソフトウェア生産技術の規律・統制である．この業務機能は，大きく，内部での定着化および，知的財産権の適切な制御からなる．

（ａ）内部での定着化

ソフトウェア生産技術の組織化で述べた，共同化，定量化，標準化，共通化の取り組みによって，規格，共通部品，共通ツール等ができても，それが，必要なソフトウェア開発プロジェクトで必要なときに使用されなければ意味はない．このため，組織内で，ソフトウ

ェア生産技術が定着するような活動が必要となる．表 2- 11 にソフトウェア開発組織において，ソフトウェア生産技術の定着化に良く使われる活動を示す．

表 2- 11 ソフトウェア生産技術の定着化活動

大分類	定着化活動	説明
教育	新人教育	組織に入る技術者全員に対して必ず実施する教育
	技術教育	新人教育より高度な内容のソフトウェア生産技術に対する教育．職種によって一部異なった内容になる場合がある
	個別教育	個別のプロジェクトの特性や課題に合わせた教育
ボトムアップな導入支援	小グループ活動	組織の TQC 活動と連動したソフトウェア生産技術の普及，改善提案等
	導入支援	プロジェクト固有の特性に合わせたソフトウェア生産技術の導入支援
トップダウンな導入支援	方針管理	組織目標の一つとしてバランススコアカード等の手法で組織トップからトップダウンに展開
	通達等	ソフトウェア生産技術部署からの通知で必要なソフトウェア生産技術の使用を徹底

（ｂ）知的財産権の適切な制御

この業務機能はソフトウェア開発組織が，ソフトウェア生産技術面で法令や他社との契約を遵守し，企業としての社会的な責任を果たすための活動である．本項では，特にソフトウェア生産技術面で課題となる組織化と規律・統制の関係をまとめる．

ソフトウェア生産技術の組織化を進めることにより，組織全体の効率は向上する．しかし，ソフトウェアの開発では，組織外の持つ知的財産を含む多くの素材を扱う．このため，単に，法律や条令を守るだけでなく，素材持つさまざまなライセンス条件を満たす必要がある．さらに，不適切な情報がソフトウェアに混入したときのソフトウェア開発組織のリスクが他の産業に比べても大きいという問題がある．表 2- 12 に知的財産権の保護のためのソフトウェア生産技術の施策を示す．

ソフトウェア生産技術の側面では，組織での全体としての効率化を優先する場面と，知的財産権に関する組織リスクを最小化する場面を両立させる必要がある．このため，ソフトウェア開発組織で，多くのソフトウェア開発プロジェクトを運用する場合，プロジェクト間で，知識を積極的に流通させる場面と，適切に知識を制御する場面があり，どのように組み合わせるかが課題となる．本件については，本論文の五章で詳細に述べる．

表 2- 12 知的財産権の保護のための施策一覧

分類	施策
条令，契約，社会的責任に対応した規律・統制	開発エリアの分離
	契約違反のソース混入防止
リスク削減のための規律・統制 [6]	ソフトウェア開発プロセスの中．複数プロジェクト間でのリスク低減
	危険なツールの統制
	営業秘密に相当するソフトウェアの流出防止
	組織内外，又は組織内でのファイヤウォール，ネットワーク分離
	セキュリティリスク削減

5. まとめ

本章では，ソフトウェア生産技術の形式知化の枠組みを提案した．まず，ソフトウェア生産技術という用語を定義し，それに関連する，過去の研究を，ソフトウェア工学誕生以前の経営工学まで遡って振り返った．この結果，過去の多くのソフトウェア工学の組織的な知識や，現在，ソフトウェア工学のトピックであるアジャイルソフトウェア開発手法も経営工学に由来した知識を活用していることを示すとともに，現状の課題として，ソフトウェア開発組織の知識という観点で経営工学とソフトウェア工学の従来の知識を統合体系化と，多様化する環境（開発技術変化，開発プロセス多様化，法的な問題）に対応した組織的な開発の確立の二点を示した．

続いて，経営工学でのモデルを利用した，ソフトウェア開発組織におけるソフトウェア生産技術のモデルを示し，その中の「ソフトウェア生産技術の対象」，「ソフトウェア生産技術の業務機能」の内容を説明した．すなわち，ソフトウェア生産技術の対象として，ソフトウェア開発者，成果物，開発支援，開発プロセス，知識を挙げ，ソフトウェア生産技術の業務機能として，組織化，継続的改善，規律・統制を説明した．

本章で説明した，ソフトウェア生産技術の定義，モデルは，従来の，経営工学やソフトウェア工学の潮流や，現状の課題を考慮したモデルとした．また，特定のソフトウェア開発組織の体制や事業目標などに依存しないモデルになっているため，本研究の適用範囲である，ソフトウェア開発プロセスを持つソフトウェア開発組織であれば，どの組織でも利用可能な知識とすることができた．

次章以降は，本章で説明したソフトウェア生産技術の体系に従ってソフトウェア生産技術の業務機能を改善した事例を説明し，本章で述べた課題の具体的な解決事例を示すとと

[6] リスク削減のための規律・統制の多くは，情報管理部署のタスクであるが，情報管理での方針に従ってソフトウェア生産技術面での施策を講じる必要がある．

もに，本章で述べた体系及び改善モデルの有効性を検証する．三章では，ソフトウェア生産技術の業務機能に対応した実際の活動事例を説明し，四章では，経営工学的な観点での定量化のモデルおよび事例を示し，五章は，知識管理的な観点での開発環境の改善および統制の事例を示す.

第三章　ソフトウェア生産技術の実用例

本章では，二章で体系化したソフトウェア生産技術が，実際のソフトウェア開発組織の抱える問題に対してどのように機能するかを述べ，その実用性を示す．

本研究で示すソフトウェア生産技術の体系は，実際のソフトウェア開発組織全体での生産性や品質を向上させるために使用できることを意図している．しかし，第二章での説明では，実際のソフトウェア開発組織での課題に対応しているのか，また，それらの課題の解決のために有効に作用するかということは必ずしも明確ではない．本章では，二章でのソフトウェア生産技術が実際のソフトウェア開発組織でどのような課題に対して，どのように使用可能なのかを以下の構成で説明する．

まず，この章以降の適用事例の対象となる日立ソフトウェア事業部の概要を紹介し，大規模ソフトウェア開発組織に一般的に見られるプロセス構成および，その中でソフトウェア開発プロセスがどのような位置づけにあるのかを示す．次に，個々のソフトウェア開発プロジェクトでの課題ではなく，大規模ソフトウェア開発組織でソフトウェア開発を行うときに頻発する組織的な課題の典型例を示す．最後に，それぞれの課題に対して，二章で示した，ソフトウェア生産技術の業務機能に従い，どのように，ソフトウェア生産技術の対象を改善，組織化，規律・統制をしているかを示し，本研究で体系化したソフトウェア生産技術が現実のソフトウェア開発組織で実用できることを示す．

1.　大規模ソフトウェア開発組織のソフトウェア開発プロセス

大規模ソフトウェア開発組織の事例として，筆者が 1980 年以降勤務している(株)日立製作所ソフトウェア事業部（以下，日立ソフトウェア事業部)の概要及び，事業部内にどのような業務プロセスがあり，各業務プロセスがどのように関連しているのかを示す．

1.1.　概要

日立ソフトウェア事業部は，(株)日立製作所情報・通信システム社の中核事業部の一つであり，パッケージ型のソフトウェア製品を主に開発する事業部である．主力ソフトウェア製品は，運用管理ソフトウェア，データベース管理ソフトウェア，アプリケーションフレームワークといった，企業の情報システムで用いられるミドルウェアで，ソフトウェア開発に携わる技術者を約 4000 名擁する． 1970 年代から事業所レベルでソフトウェア開発の標準規格を作り組織的なソフトウェア開発を推進している．

1.2.　大規模ソフトウェア開発組織のプロセス構成

日立ソフトウェア事業部を例にとり，大規模ソフトウェア開発組織のプロセス構成と，その中でのソフトウェア開発プロセスの位置づけを示す．

ソフトウェア開発を目的とする組織においては，ソフトウェア開発プロセスがその組織の基幹プロセスの中でもっとも重要なプロセスである．しかし，大規模ソフトウェア開発組織においては，通常，ソフトウェア開発プロセス以外の業務プロセスを持っている．例えば，日立ソフトウェア事業部のように，ソフトウェアを製品として市場に出荷するような組織の場合，ソフトウェア開発プロセス以外にも，図 3- 1 が示すように組織の業務プロセスとして，製品企画プロセス，出荷プロセス，拡販プロセスがある．

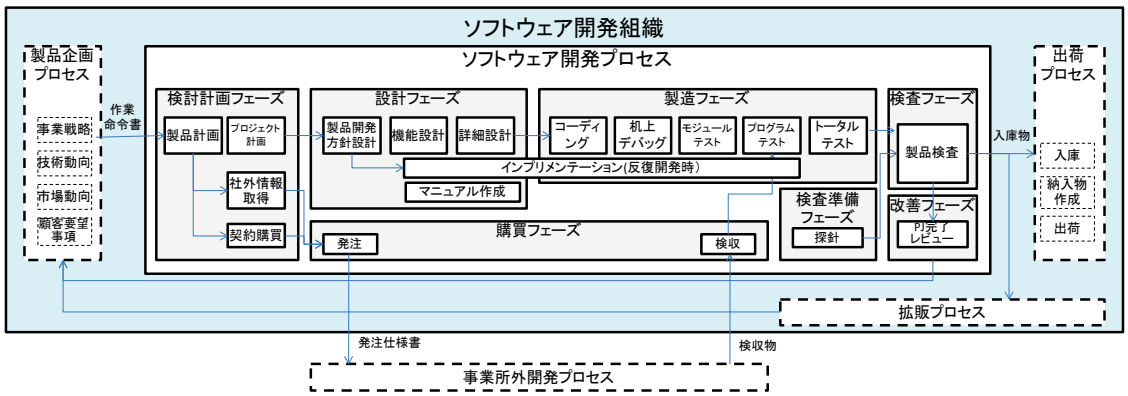

図 3- 1 ソフトウェア製品開発組織の典型的なプロセス構成

これらのプロセスごとにそのプロセスを主管する部署が分かれており，企画担当部署，ソフトウェアの製品開発部署，ソフトウェア検査部署，出荷担当部署，拡販担当部署がそれぞれの(サブ)プロセスの責任を負う．日立ソフトウェア事業部の場合，メインとなるソフトウェア開発プロセスおよび，他のプロセスとのインタフェースは事業所の規格で決められており，また，常時多数のプロジェクトが並行して実行されている．このような組織では，ソフトウェア開発プロセスを固定的な生産システム，その中で開発されるソフトウェアを成果物としてモデル化可能である．すなわち，図 3- 2 が示すように，ソフトウェア開発組織において，ソフトウェア開発プロセスが，企画プロセスからの入力，すなわち要求を常時処理し，出荷，拡販といったソフトウェア提供のための後置プロセスに成果物を出力するような生産システムとみなすことができる．

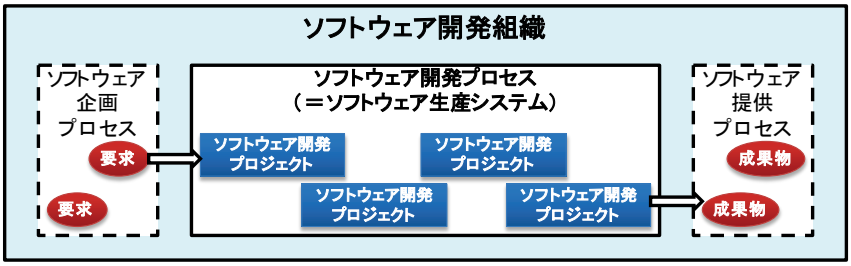

図 3- 2 ソフトウェア生産システムとしてのソフトウェア開発プロセス

このようにソフトウェア開発プロセスを生産システムとしてモデル化した場合，その生産システムの効率向上が組織的な目標の一つとなる．すなわち，そのシステムを通過する，一つ一つの成果物の開発過程の効率化だけでなく，生産システムとしてのソフトウェア開発プロセス全体での効率向上が必要である．さらにソフトウェア開発プロセスだけでなく，他のプロセスと連携した組織的な効率向上も組織全体として重要となる．

なお，大規模ソフトウェア開発組織であってもハードウェア製品の組み込みソフトウェアを開発する組織や，エンタープライズ系の受注系のソフトウェアを開発する組織では，ソフトウェア開発プロセスを取り巻く各プロセスの構成やプロセス間のインタフェースは異なってくる．しかし，ソフトウェア開発プロセス部分が生産システムとしてモデル化できること．他の業務プロセスがソフトウェア開発プロセスと関係している点は，どの大規模ソフトウェア開発組織でも共通している．

2. 組織的なソフトウェア開発での典型的な課題

本節では，ソフトウェア開発組織が定常的な生産システムとしてのソフトウェア開発プロセスを構築し，運用するときに，良く発生する課題の例を示し，二章で示したモデルによってどのように改善，組織化，規律・統制ができるかを，日立ソフトウェア事業部での事例を使いながら説明する．

個々のソフトウェア開発プロジェクトではなく，それを定常的に開発し続ける組織をソフトウェア生産システムとみなしたときの典型的な課題の例を示す．ここに挙げる例がすべての課題ではないが，本章の目的である，ソフトウェア生産技術モデルの実用性を示すために，多くの組織で共通に問題なる可能性の高い課題を抽出した．

課題１　プロジェクト単位で採取する各種データの組織化

組織で定量的に多数のソフトウェア開発プロジェクトから採取したデータを組織全体の数値として集計可能にする必要がある．ソフトウェア開発組織の場合，各ソフトウェア開発プロジェクトの多様性が大きく，単にプロジェクトのデータを集計しただけでは組織的な実態把握および改善にならない場合が多い．

課題２　プロジェクト単位の効率追求より組織全体の効率化

ソフトウェア開発プロジェクトそれぞれが，個別に効率化を進めた場合，必ずしもソフトウェア開発組織全体の効率向上につながらない場合がある．例えば，ソフトウェアの再利用を考えた場合，ソフトウェア開発プロジェクトの生産性を考えた場合，再利用を行わないほうが，そのプロジェクトの生産性は向上する．しかし，ソフトウェア開発組織の観点からは再利用を進めたほうが生産性の向上が観測できるという問題がある．

課題３　各部署，各ソフトウェア開発者の持つ知識の集積，共有，組織的な知識の創成

ソフトウェア開発組織が組織的に開発することを強みにするためには，その組織固有の知的資産を蓄積することが必要である．再利用可能なソフトウェア資産の蓄積は重要である．また，ソフトウェアそのものでなくても，開発支援ツール，ソフトウェア開発に関する知識，ソフトウェア対象ドメインに対する知識などを，各個人，各プロジェクトだけでなく，組織全体で集積，共有，創成できる必要がある．また，知的財産権の問題などで，組織的に共有することに対するリスクへの配慮も必要となる

課題４　組織レベルでの共通化による各種管理負荷の効率化

各ソフトウェア開発プロジェクトでは，そのプロジェクト関連の各管理（プロジェクト管理，品質管理，構成管理等）を行う必要がある．管理自体は必要不可欠であるが，実際の管理負荷の多くは，管理システムの保守（計算機の管理，バックアップ等の作業等）や，管理システムの構築やトラブル対応の場合が多い．このようなプロジェクト横断的な管理負荷を組織全体で効率化する必要がある．

課題５　各種組織施策の組織内への徹底

ソフトウェア開発組織で，その目標，施策，知識などをいかに組織内のソフトウェア開発プロジェクトやソフトウェア開発者に徹底するかが問題となる．

3. ソフトウェアの組織的開発の実例

本節では，前節で示したソフトウェア開発に関連する組織的な課題に対して，どのように二章のモデルを使って改善，組織化，規律・統制をしていくかを，日立ソフトウェア事業部の実例を通して説明する．

3.1. 継続的改善の事例１：プロジェクト単位での採取データの組織化

本項は，前節で示した課題のうち，「課題１ プロジェクト単位で採取する各種データの組織化」に対して，二章で示したソフトウェア生産技術の改善の STEP1 において，組織化の考えを使って解決した例を示す．

ソフトウェア開発組織で組織全体の継続的な改善を行うためには，その前提として，QCD のレベルの基本的なデータ項目を組織的に採取できるようにして，生産性，品質密度等の基本的な項目による組織全体の定量測定が必要である．

本項では，単にソフトウェア開発プロジェクトにおいてその基本的なデータを採取可能にした事例ではなく，多数のソフトウェア開発プロジェクトから採取したデータを組織全体の数値として集計可能にするために，どのような課題があり，どのように解決するのかを日立ソフトウェア事業部の事例を使って述べる．本項の詳細は，芝田の解説 [65]，居駒の論文 [66]を参照のこと．

（1）組織レベルの定量化の課題

組織的なデータ活用のためには，開発プロジェクトの多様性の問題を解決する必要がある．ソフトウェア開発の場合，個々のプロジェクトの開発形態が，それぞれ，他のプロジェクトと大きく異なっている．まず，個々のプロジェクトが独立に採取基準を決めた場合，組織としての集計は困難となる．さらに，同じデータを単に複数プロジェクトで集計したり統計的な数値を求めたりしても，ばらつきが大きすぎて組織として意味のある数値にならないことが多い．

（2）プロジェクトの多様性の課題に対する施策

ソフトウェア開発プロジェクトの多様性により，組織的な定量化が困難になる課題に対する施策は，大きく，同一のメトリクス使用とプロジェクト特性による正規化の二つであり，これを組み合わせることによって基準値をソフトウェア開発にも導入した．

(a) 同一メトリクスの使用

ソフトウェア開発においては，開発するソフトウェアの種類や目的によって，本質的に開発プロセスが異なってくる場合が多い．一方，ソフトウェア開発の工数や生産量，ソフトウェアフォールトの分類など，組織全体で統一できるものも少なくない．日立ソフトウェア事業部においては，開発工程およびその(中間)成果物としてのドキュメント体系，形式，および記述方法を規格として標準化を進めた [65].

(b) 正規化

開発プロセスを標準化しても，その結果として得られるデータは個々のプロジェクトの特性によって大きな分散を持つ．難易度の高いオペレーションシステム（OS）に近いソフトウェアを開発する場合，生産性が，アプリケーションプログラムより低くなることはよく知られている [32] [66]．また，品質面でも，例えば，ソフトウェアの種類と出荷後にフィールドで摘出されたフォールト全体の数を比べると，図 3- 3 のように，OS のフォールト数は，言語プロセサのフォールト数に比べて 30%以上多い [66].

また，開発するプロジェクトの規模によっても，成果物としてのソフトウェアの品質は大きく変わってくる．図 3- 4 のように管理体制の違いによってソフトウェア開発プロジェクトの規模を分類したときに，図 3- 5 に示すように一人の管理者が一つのプロジェクトに専任できるような中規模のソフトウェア開発プロジェクトの成果物の品質が，管理者が他のプロジェクトを掛け持ちしていたり，複数の管理者が階層的に管理したりしているプロジェクトよりも成果物の品質が良いことが分かっている．

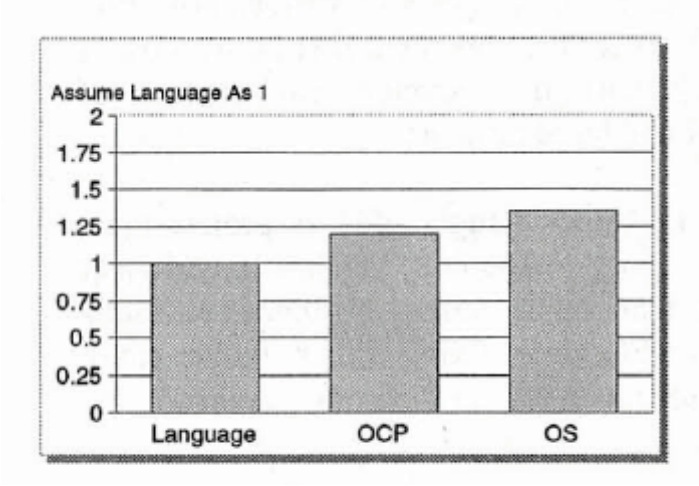

図 3- 3 ソフトウェア製品の種類と出荷後フォールト数の関係

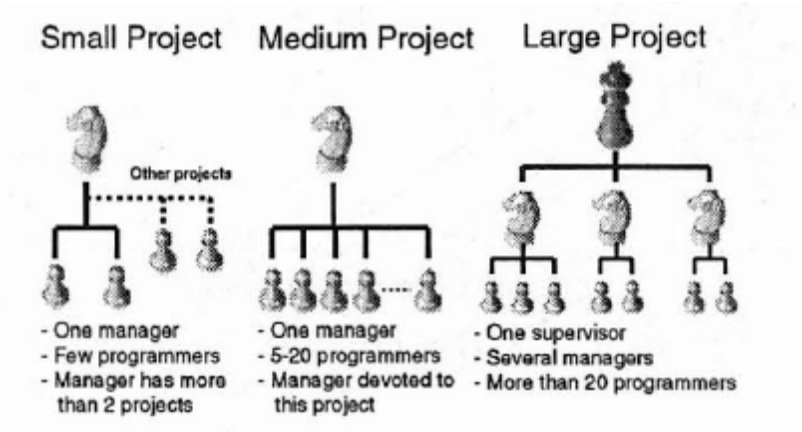

図 3- 4 ソフトウェア開発プロジェクト規模の分類

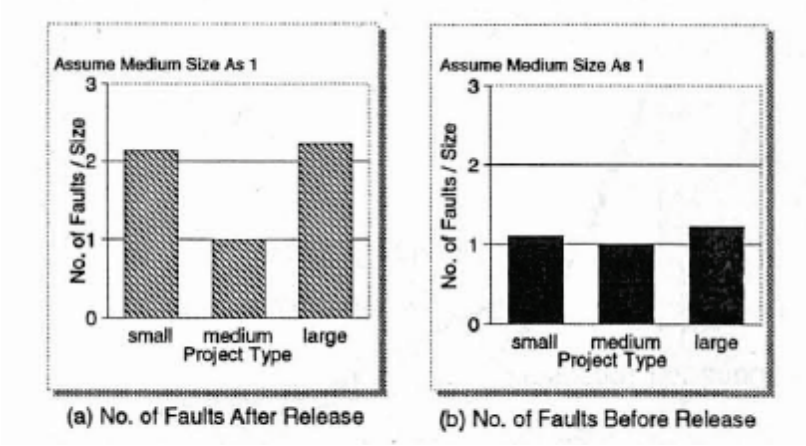

図 3- 5 プロジェクト規模と出荷後のフォールト数との関連

これらのデータを機械的に統計処理した場合，分散が大きくなるだけでなく，計測時の計測対象の製品群のうち，特長的な製品の種別によって組織全体の値に大きな差が出てくる場合がある．このため，日立ソフトウェア事業部では，各ソフトウェア開発プロジェクトの特性を下記のように決めた．

- 対象となるプラットフォーム
- プログラム種別
- 開発の新規性（難易度）
- 使用言語
- 開発者のスキル[7]

これらの特性により，ソフトウェア開発プロジェクトの生産性がどのように影響を与えるかを過去のソフトウェア開発プロジェクトの結果から多変量解析を行った．このようにして求めた係数によって，計測対象の多数プロジェクトの正規化を行ったうえで組織全体の集計を行っている [65] [67].

(c) 基準値の導入

上述の標準化と正規化を組み合わせ，ハードウェア製造組織で広く用いられている基準値の考え方をソフトウェア開発にも導入した．芝田 [65]によると，基準値の定義は，以下である．

「所定の作業方法のもとで，その作業について，標準的なプログラマが作業を遂行するのに必要となる基準工数および，基準計算機使用時間」

前述の通り，ソフトウェア開発においては個々のプロジェクトの特性によって大きな分散を持つ．まず，過去のソフトウェア開発プロジェクトから同じ基準で採取したデータを多変量解析し，図 3- 6 に示すような基準値の構造を事業所全体の仕組みとして構築した．

[7] 芝田の解説では開発担当者のスキルは見積もり時に特定困難なことを理由に特性に入れていないと書かれているが，その後導入されている．

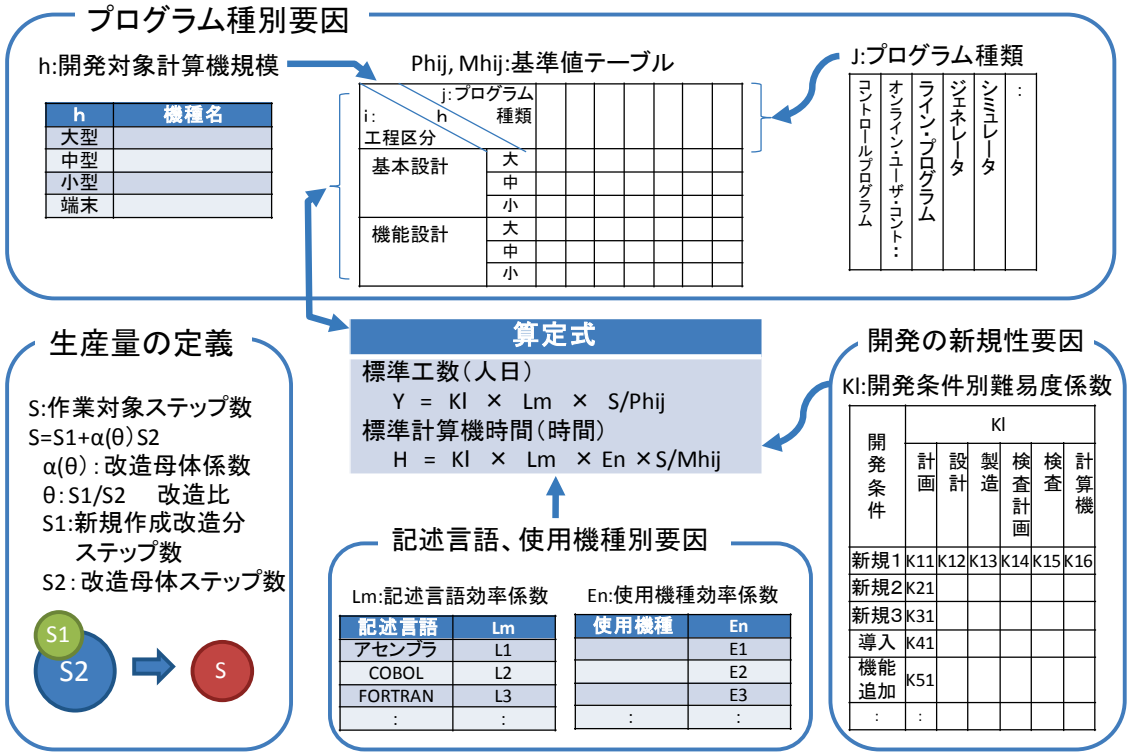

図 3- 6 日立ソフトウェア事業部における基準値の構成と要因（芝田 [65]）

3.2. 継続的改善の事例２：開発効率の導入

本項は，前節で示した課題のうち，「課題２　プロジェクト単位の効率追求より組織全体の効率化」と「課題３　各部署，各ソフトウェア開発者の持つ知識の集積，共有，組織的な知識の創成」に対して，二章で示したソフトウェア生産技術の改善の STEP2 から STEP5 の考えを使って解決した例を示す．

本事例は，ソフトウェアの再利用や，マルチプラットフォーム対応のソフトウェア開発時の事業課題から，メトリクス設定，PDCA による管理，メトリクスの定期的な見直しを行った事例を紹介する．以下，STEP2-5 の各ステップに従って説明する．

（1） STEP２　改善目標の設定

1970 年代～1980 年代までは，ソフトウェアの開発対象プラットフォームのほとんどは，メインフレームコンピュータで，複数のプラットフォームで動作するソフトウェアの開発は皆無に近かった．1990 年代以降，Windows プラットフォームおよび各社提供する UNIX プラットフォームが普及し，一つのソフトウェア製品が，複数のプラットフォームをサポートする事例が増加してきた．このため，事業目標として，「ソフトウェア製品のオープンプラットフォーム化」が設定され，その目標を効率的に達成するために，生産技術上の目標として以下の二点を設定した．

- ソフトウェア製品の効率的なマルチプラットフォームサポート
- 再利用ライブラリの創成および，ソフトウェア製品の（他社製を含めた）再利用可能ライブラリの積極的活用

（2） STEP３　メトリクス設定

STEP2 で設定した目標に対応し，事業部内で推進するメトリクスを設定した事例を紹介する．

(a) 課題

まず，目標を達成するために，現状の仕事の設定方法，メトリクスなどにどのような問題があるのかを洗い出した．それまでの生産性は，次の式 3-1 で計測していた．

$$\text{(狭義の)生産性} = \frac{\text{作業対象ステップ}}{\text{総原価}} \qquad (3\text{-}1)$$

ここで，作業対象ステップとは，新規に加えたステップ数に改造の母体になる規模に係数をかけて加えたものである（図 3- 6 の左下の図参照）．この定義の生産性は「狭義の生産性」または，作業効率と呼ばれ，同じソフトウェア開発をしたときのソフトウェア開発者+ソフトウェア環境の効率を計るメトリクスである．このメトリクスを使ったときに以下のような課題があることを識別した．

- **再利用の問題**

個々のソフトウェア開発プロジェクトが同じようなソフトウェアを開発したとき，見掛け上の生産性は向上するが，組織全体としての効率は同じソフトウェアを再利用したときに比べて低下する．また，ソフトウェア開発組織が組織的に開発することを強みにするためには，その組織固有の再利用可能なソフトウェア資産を持つことが重要である．ところが，あるソフトウェアを開発時，必要な機能を実装するときに，既存のソフトウェアを再利用すると，ソフトウェア開発組織の観点からは生産性が向上するが，ソフトウェア開発プロジェクトの生産量は減少し生産性は低下するように見えるという問題があった．

また，再利用を目的としたソフトウェア部品を作る場合も，多くのソフトウェアで再利用可能なように汎用的に部品を作るほうが組織的には効率的であるが，汎用的に開発するほうが開発の難易度は高くなり，その結果その部品の開発コストは増加し，部品開発プロジェクトの作業効率は低下するように計測されてしまうという課題がある．

● マルチプラットフォームの問題

1990 年代までのマルチプラットフォームに対する日立ソフトウェア事業部の開発プロセスは，図 3- 7 の左図のように，ベースバージョンの開発プロジェクトのあとに，サポートするプラットフォーム毎に移植プロジェクトを順番にシリアルにプロジェクトを設定するという手法をとっていた．このため，最初のプラットフォーム対応のソフトウェアが完成してから，最後のプラットフォーム対応のソフトウェアが完成するまでに，数か月の遅れが出ていた．一方，この分野の先進企業である，米国のX社をベンチマーキングしたところ，図 3- 7 の右図のように複数プラットフォームのサポートを並行して一つのプロジェクトとして実行し，サポートするプラットフォームの出荷時期をほぼ同時にしていた．ビジネスの観点では，X社のプロセスのほうが顧客の環境に適応したソフトウェア製品を早期に出荷できるため断然優位であるが，単に，生産性という観点で比べると差が出ないばかりか，ビジネス的に不利な自社プロセスのほうが良くなるという課題があった [8].

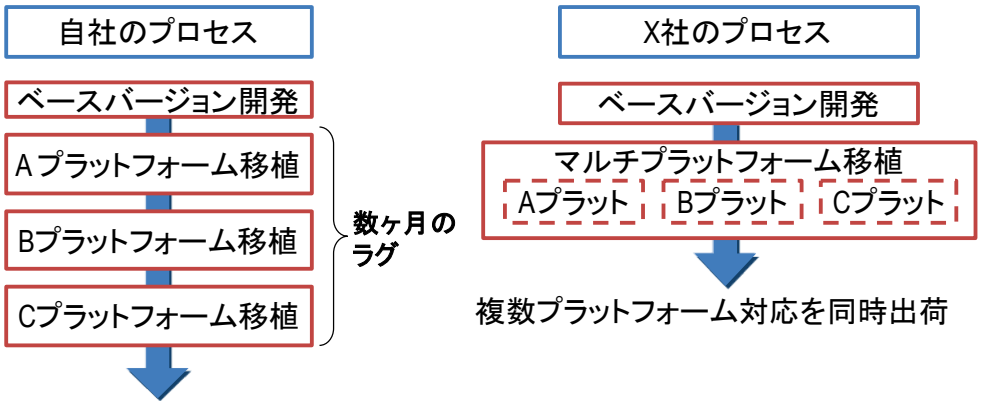

図 3- 7 複数プラットフォームサポート時のプロジェクト構成

(b) 課題解決の方向性

上述のプロジェクトと製品レベルでの効率化の矛盾は，ソフトウェア開発プロジェクトとソフトウェア製品の関係において，再利用の存在が定義されていないことによると考えた．ソフトウェアはそのライフサイクルの中で複数回，ソフトウェア開発プロジェクトを経る．ハードウェア製品と異なり，ソフトウェア製品はその生産システムの成果物として作られたものが，また，次のソフトウェア開発プロジェクトにおいて，その生産システムの入力となるからである．従って，ある時点でのソフトウェア製品は，複数のソフトウェア開発プロジェクトを介して開発されたといえる. この関係を UML のクラス図でモデル化したのが図 3- 8 である．

[8] 図 3- 7 の左図ようにプロジェクトを順々に実行するときと，右図のように一プロジェクトとして実行するときの生産性とビジネス効果の比較は第四章で詳述する．

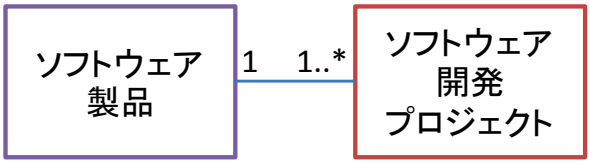

図 3- 8 一般的なソフトウェア製品と開発プロジェクトの関係モデル

あるソフトウェア製品が初期開発され，次々と機能追加や，フォールトの修正をする場合には，このモデルで適合する．しかし，実際のソフトウェア製品は上記のような単純なモデルでは表せない．ソフトウェア再利用の普及により，ある製品用に開発されたソフトウェアのコードが，他の製品に組み込まれる場合も増加している．このため，市場で稼動しているソフトウェアは，図 3- 9 のようにさまざまなソフトウェア開発プロジェクト，およびソフトウェア修正作業で開発，修正したコードから成り立っているのが現状である．ソフトウェア開発プロジェクト側からみても，ある製品用だけに開発される場合もあるが，複数の製品に利用されるようなソースコードであったり，複数のプラットフォームで動作する製品であったりする場合が増加してきた．

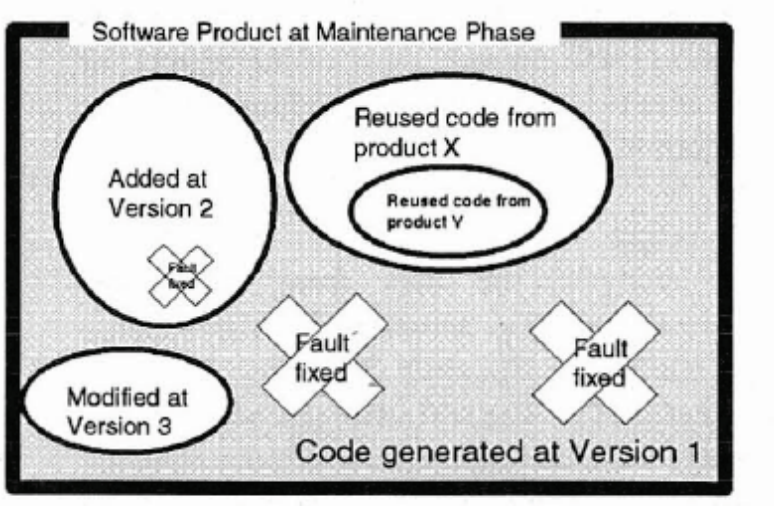

図 3- 9 稼働しているソフトウェア製品の構成例

このため，製品の構成として，ある製品に，他の製品が部品的に含まれるようなモデル，また，あるソフトウェア開発プロジェクトがあった場合，対応する製品が一つに限定されず，二製品以上に対することができるようにモデル化することが必要となる．この課題に対応したモデルを図 3- 10 の UML クラス図に示す．

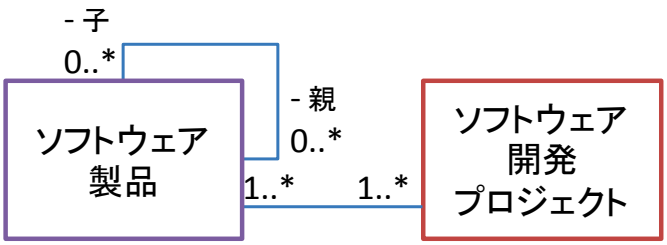

図 3- 10 構造的な製品体系や再利用プロジェクトを考慮したモデル

このモデルの場合，ソフトウェア製品は図 3- 8 の基本的な関係以外に，ソフトウェア製品自身が，構造的に他の子ソフトウェア製品を複数包含することや，あるソフトウェア開発プロジェクトが，複数の製品に対応するような関係も含んでいる．

この関係を使って，どのように，ソフトウェア開発プロジェクトの効率を計測するかを次に説明する．

(c) 開発効率メトリクスの設定

前項で説明した UML のクラス図に従って，完成したソフトウェア製品が，どのようなプロジェクト履歴を経ているかは分かり，その開発に要した原価が把握でき，広義の生産性を式 3-2 に計測できる．

$$\text{広義の生産性} = \frac{\text{総売上}}{\text{総原価}} \tag{3-2}$$

しかし，このメトリクスには二点の課題がある．第一に，図 3- 10 の UML により，一つのプロジェクトが多数の製品に使われる可能性があるため，製品に関係あるプロジェクトの原価を単純に加算したのでは過剰な原価となる．第二の課題は，広義の生産性は，総売上が計測できてから得られる結果指標であり，実際に開発したソフトウェア製品が市場に出て収益が出ないと計測困難なことである．二章の定量化の要件で示したように，組織的に使用するソフトウェア生産技術のメトリクスとしては，結果としての指標だけでなく，推進目標となるような指標，推進指標が必要である．

第一の課題は，ソフトウェア開発プロジェクトを実行するときに，複数のソフトウェア製品の原価として按分できるように設定することで解決した．第二の課題に対しては，片岡 [68]の生産性の考え方に基づき，式 3-3 のような生産性指標の構造を導入することで解決した．

$$\text{広義の生産性} = \frac{\text{総売上}}{\text{総原価}} = \frac{\text{作業対象ステップ}}{\text{総原価}} \cdot \frac{\text{開発ステップ}}{\text{作業対象ステップ}} \cdot \frac{\text{総売上}}{\text{開発ステップ}} \tag{3-3}$$

すなわち，作業ステップと，結果としての総売上の間に開発ステップという概念を導入して，ここで，再利用率やマルチプラットフォームサポートの要因を開発規模に反映できるようにした．ここで，開発ステップとは，

$$\begin{aligned}&\text{開発ステップ}\\&= \text{作業対象ステップ}\\&\times \theta\left(\text{サポートプラットフォーム数}\right) + \sum \text{再利用部品毎の認定ステップ}\end{aligned} \tag{3-4}$$

とした．すなわち，図 3- 6 で説明した作業対象ステップに，その開発プロジェクトで，どのようなライブラリを使ったか，また，成果物のソフトウェア製品は，どれだけのプラットフォームで動作するかという要因を加えた．この開発ステップを使い，開発効率を，式 3-5 のように定義した．

$$\text{開発効率} = \frac{\text{開発ステップ}}{\text{総原価}} \tag{3-5}$$

この開発効率により，ソフトウェア開発者に対して，生産性を損ねることなく再利用部品を活用させることが可能になり，さらに，マルチプラットフォーム対応のソフトウェア製品を作るソフトウェア設計者に対して図 3- 7 の右側のような適切なプロセスを採用するモチベーションを与える．また，複数ソフトウェア製品で使用されるような部品を開発するプロジェクトの原価を按分できるようにすることにより，部品を開発するインセンティブを組織的に与えることを可能にした．

（3） STEP4　設定したメトリクスによる継続的改善

前項で説明した開発効率メトリクスは，試行期間を経て係数等を確定させその後は，事業部で開発される全ソフトウェア製品，全ソフトウェア開発プロジェクトに適用している．さらに，事業部全体及び，部署毎に目標管理を行っている．すなわち，年に二回，事業部全体および各部署の目標を立て，その目標に沿って開発効率を管理し，開発効率の悪い部署およびプロジェクトに対しては，原因の追究および問題の是正を図っている．

また，開発効率を算出するために必要最小限のパラメタのみをソフトウェア開発者が入れれば，自動的に開発ステップを算出するツールを開発し開発者の計測負荷を軽くした．

（4） STEP5　定期的なメトリクスのスクリーニング

開発効率メトリクスは，年に二回見直しを行っている．見直しでは，再利用ライブラリの認定や，再利用時の係数の見直しを行い，その時点での事業部の実態に合うメトリクスであるようにしている．

3.3. 組織化の事例　構成管理システムの組織共通化

本項は，前節で示した課題のうち，「課題４　組織レベルでの共通化による各種管理負荷の効率化」に対して，二章で示したソフトウェア生産技術の組織化に基づき，ソフトウェア開発ドキュメントやソースコードの構成管理システムを組織内で共通化した事例を取り上げる．

（1） 課題

日立ソフトウェア事業部では 1970 年代から，組織全体でソフトウェアの変更管理の規格を持ち，すべてのソフトウェア開発プロジェクトは規格に沿って，機能追加や，ソフトウェアフォールトの修正といったソフトウェアの変更を行っている．しかし，2001 年時点では，ソフトウェアのソースや，ドキュメントといった変更管理の対象となるソフトウェア構成項目(software configuration item)としての(中間)成果物をどのようなツールでどのような形式で管理するかはプロジェクト毎に個別に効率化を図っていた．この時点での構成を図 3- 11 に示す．二章で説明した組織化の用語を使うと，標準化まではできていたが，共通化ができておらず，非効率的な業務機能になっていた．このため，プロジェクトの中では効率化を図ったつもりでも事業所全体で考えると，下記のような三つの課題があった．

- ソース管理の問題

ソースの構成管理は電子的に行っていたが，構成管理システムはプロジェクト毎にサーバを立ち上げ，ソフトウェア開発者が運用管理していた．従って，高価なサーバマシンが多数存在するだけでなく，構成管理ツールの保守やバックアップといった管理作業に，優秀な開発者の負荷が取られるという問題もあった．さらに，適切なバックアップ作業を採らずに，ハードウェアトラブルで数日分の開発作業が無駄になるというような事例もあった．

- 提供媒体作成の問題

この時点で，ソフトウェア生産プロセスの次プロセスである，出荷プロセスは自動化されており出荷システムの要求する形式の媒体を出荷システムに入庫すれば，発注した顧客

に対してソフトウェア設計者の作業を介さずに出荷できる仕掛けはできていた．しかし，ソース構成管理の仕掛けがプロジェクト毎にばらばらだったため，出荷システムに入庫する媒体を作成する作業がプロジェクト毎にばらばらになり，この作成工数が増えるという問題があった．

- ドキュメント管理の問題

ドキュメント自体はワードプロセッサを使用し電子形式で作成されるが，その管理方法はプロジェクトでまちまちであった．また，開発規格で決められた審査・承認処理を電子的に行うシステムは普及しておらず，構成管理対象のドキュメントの原本は紙ベースで，変更管理や，入庫等の作業は，手作業で行っていた．

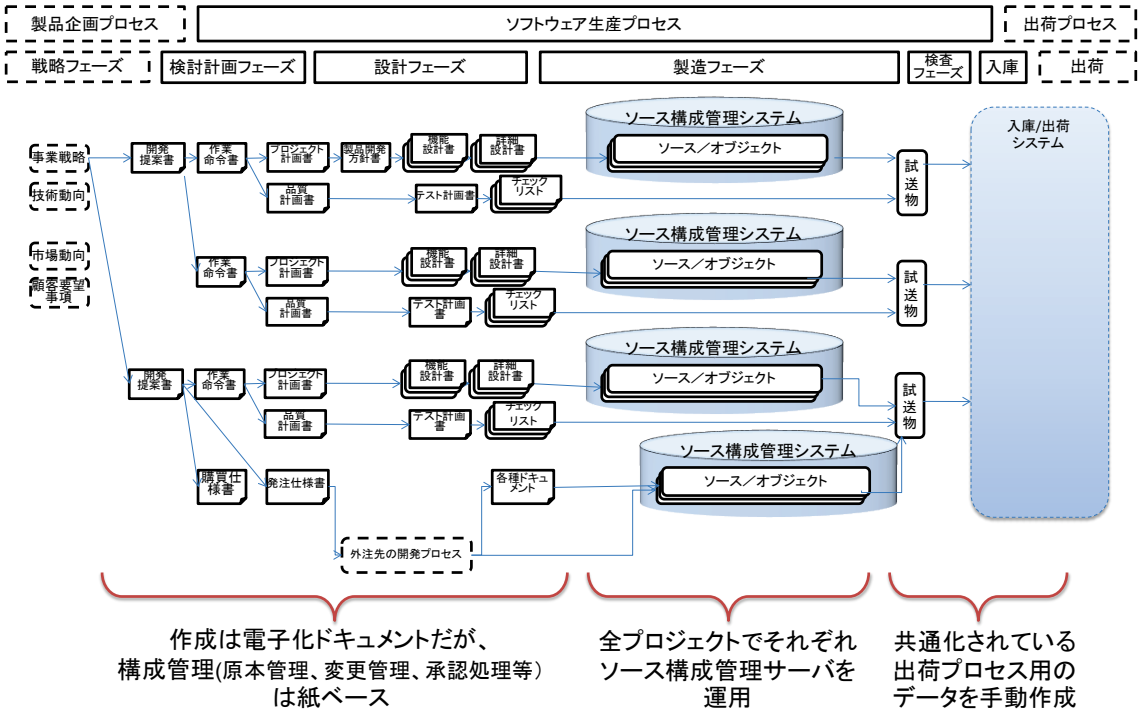

図 3- 11 組織共通化する前の構成管理環境

（2） 事業所レベルの構成管理システムの構築

前項で示した課題に対応するため，事業部全体のプロジェクトで活用可能な構成管理システムを構築した．構成管理システムの概要を図 3- 12 に示す．この図で明らかなように，検討計画フェーズから設計フェーズでソフトウェア開発プロジェクトが，各々管理していたドキュメントや，製造フェーズでのソースコードを，事業所全体の構成管理システムとして，大きくドキュメント向けの構成管理システムとソースコード向けの構成管理システムの二つに集約した．

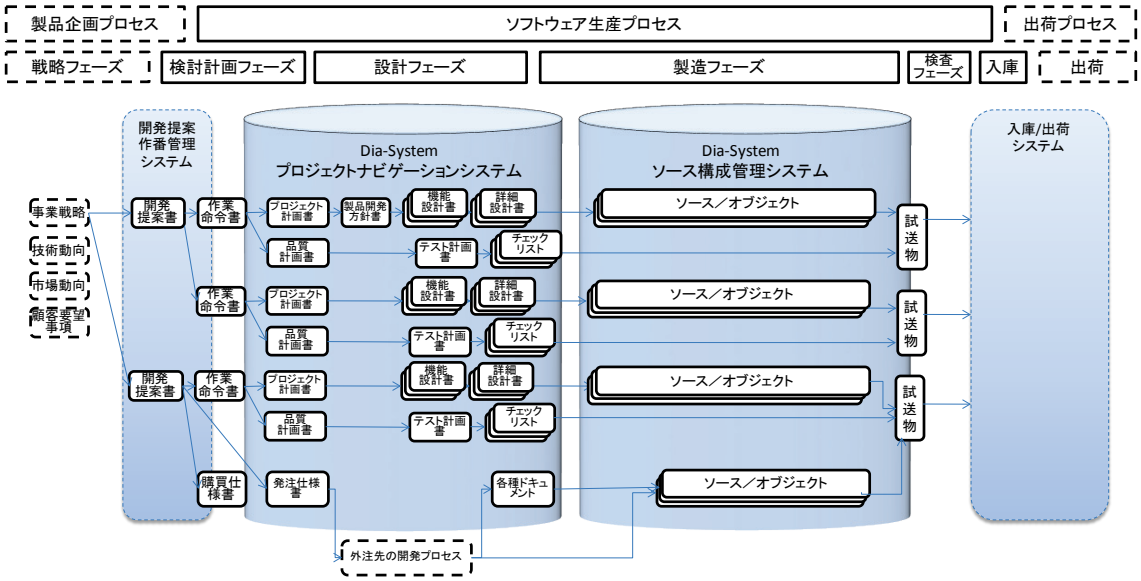

図 3- 12 組織共通化した構成管理環境

このシステムおよびその運用の特徴，組織化の効果は下記である．

- 電子的な成果物管理システム

事業部全体で活用可能な成果物管理システムを構築した．このシステムは，単に，成果物を電子的に管理するだけでなく，開発部署での審査・承認処理の他，品質保証部署のドキュメント検査にも対応している．Web ベースの画面で操作可能なため，ソフトウェア開発プロジェクト毎の導入負荷はほとんど無い．

- 事業部ベースのソース管理システム

成果物管理システムと同様にソースの構成管理システムも事業部全体で共有するようにした．各ソフトウェア開発者に対しては，構成管理のクライアント（Windows アプリケーション）を配布した．海外を含むソフトウェア発注先の組織には，発注先のサイトに同種の構成管理システムを設置し，日立ソフトウェア事業部の構成管理システムと一日一回の単位でソースコードの同期がとれるようにした．

- 入庫物，提供媒体の自動生成機能

電子的な成果物管理システムで作成されたドキュメントと事業部全体のソース管理システムで開発したソフトウェア製品を統合し，プロジェクトが終了時に，出荷システムに成果物をまとめた入庫物と顧客への提供媒体をディスク上で仮想化されたイメージで生成する機能を設けた．この機能により，これまで，各ソフトウェア開発プロジェクトで実施していた提供媒体作成作業が抜本的に低減した．

● 高信頼システムの構築

上記に説明した成果物管理システムおよび事業部ベースのソース管理システムでは，コンテンツの格納は RAID システムで，毎日，無停止で自動的にバックアップを採取するしかけを構築し，ソフトウェア開発者の管理負荷低減に加え，データの保全性も確保した．

● 適用支援および適用率管理

これらのシステムを運用開始後，日立ソフトウェア事業部の各製品は，基本的に新しいプロジェクトから順次，従来の構成管理システムから構築から新しいシステムに移行した．ただし，従来の構成管理手順からの移行が必要な製品もあり，移行できる製品の新しいプロジェクトから順次移行を進めた．移行を推進するために，各部署，事業部全体で移行目標を立て，システムによって，3〜4 年間，適用率を計測して管理した．適用できるすべての製品が移行した時点で，適用率の計測は終了している．

3.4. 規律・統制の事例　組織知の展開事例

本項は，前節で示した課題のうち，「課題５ 各種組織施策の組織内への徹底」に対して，二章で示したソフトウェア生産技術の規律・統制に基づき，組織化の事例で示した事業部レベルの構成管理システムを題材にどのように，規律・統制を行ったかの事例を紹介する．

（１） 課題

形式知として組織全体で標準化されても，例えば，単に Web に格納しているだけでは，実際のソフトウェア開発プロジェクトで活用されない場合がある．また，その知識を必要とするソフトウェア開発者が，必要なソフトウェア開発プロジェクトの工程で見逃される場合も少なくない．

（２） 非技術的な規律・統制の事例

構成管理システムの説明会を全部署対象に実施したほか，新しい構成管理システムの操作や構成管理システムを使ったコンテンツの管理方法等の教育を実施した．また，各部署の方針管理の仕掛けを活用し，各組織の期の方針（年二回）で，新システムの活用を事業部幹部にコミットするようにした．

また，組織の全員運動，小集団活動などのボトムアップ活動と連携し，ソフトウェア開発者のレベルから自発的に使用を開始するようにした．また，ソフトウェア開発者からのクレームや要望は，Web ベースの電子会議室経由で，システム運用者と会話ができるようにして，使用者の生の声をクイックレスポンスできるようにした．

（３） 技術的な規律･統制の例

日立ソフトウェア事業部がその長い歴史で蓄積した形式的な知識は膨大な量に上る．ソ

フトウェア開発プロジェクトは必ず遵守しなければならない規格レベルの知識もあれば，知っておけばさらに良いというレベルの知識もある．知識の一部は，20 年近く前に確立したメインフレーム用の知識であるし，これから開発するプロジェクトにしか役に立たない知識もあるかもしれない．また，ソフトウェア製品企画段階の知識もあれば，ソフトウェアテストの基本知識もある．こういった，多種多様な知識を，その知識を必要とするソフトウェア開発プロジェクトの必要な工程で確実に参照されるようなしかけを，図 3- 13 に示すプロジェクトナビゲーションシステムとして実装した．

左ペインは，事業部の規格に従ったソフトウェア開発プロジェクトでの標準 WBS を木構造で表示している．プロジェクトごとに標準 WBS をカスタマイズすることも可能である．この WBS の一つ一つのタスクが，ソフトウェア開発プロセスの一つの工程に対応し，その工程で作成すべき成果物が右上のペインに表示される．右上のペインに登録した成果物は，事業部の規格に従った変更管理，審査・承認を Web 上で管理可能である．さらに右下のペインでは，今表示している工程での，規格や，設計ガイド，インフォーマルな知識が表示され，このシステムを使用しているソフトウェア技術者が，使用しているときに必要とする知識を簡単に得ることができる．このシステムを使用することにより，組織化という観点で，組織での成果物管理を組織内で共通化しただけでなく，組織内で蓄積した知識をソフトウェア開発者，運用者の負荷を増加することなく，規律可能とした．

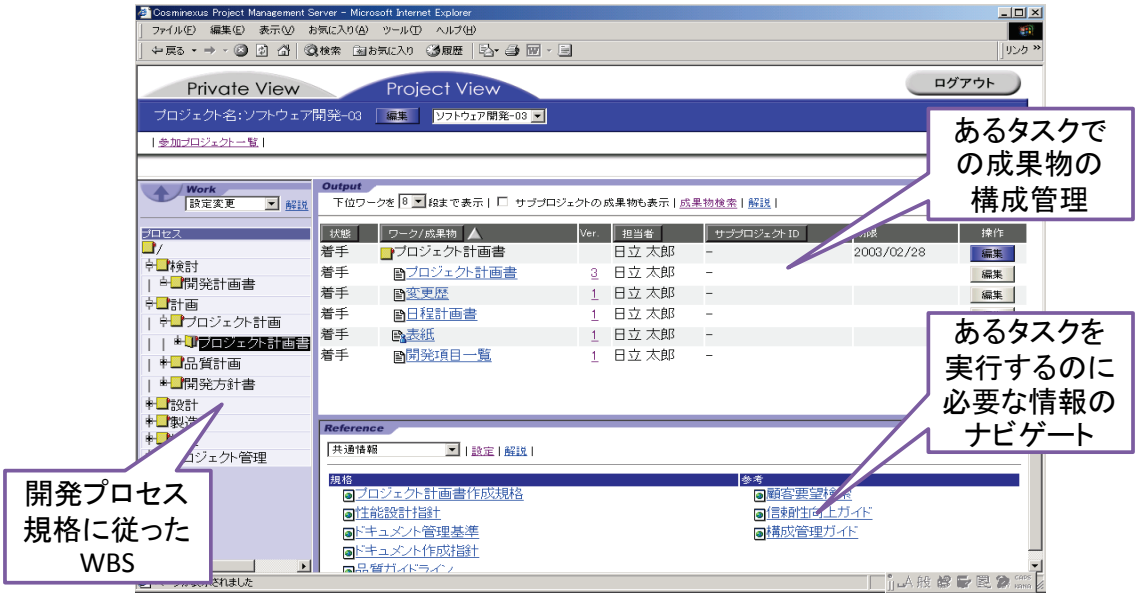

図 3- 13 プロジェクトナビゲーションシステム

さらに，本システムは，ハードウェアとしては，組織内で共通化しているがデータや使用者のビューでは，プロジェクト毎に明確に分離し，アクセス制御を行っている．このため，知識財産権の保護の課題に対しても有効なシステムとなっている．また，組織的に一

つのサーバにデータを集約したことにより，組織共通のツールを使いやすい環境になっている．例えば，開発したソフトウェアやドキュメントに不必要な他社の著作物が混入していないかといったチェックも，そのようなツールを起動すればチェック可能なような構成になっている．

4. 本章のまとめ

本章では，二章で述べたソフトウェア生産技術の対象および業務機能のモデルが，実際のソフトウェア開発組織のかかえるどのような組織的な問題に対して，どのように実用可能かを示した．

本章での事例は，ソフトウェア開発組織で生じる全ての組織的な課題に対する解決を行っているものではないし，二章でのモデルのすべての項目に対する事例を示したものでもない．しかし，大規模ソフトウェア開発組織ソフトウェア生産技術の業務機能がどのように，ソフトウェア開発組織に実装され，どのように組織的なソフトウェア開発を可能にしていることを示せたと考える．この章に続く，四章，五章では，より最近のソフトウェア開発組織の課題に対する，ソフトウェア生産技術の事例を紹介する．

第四章　大規模ソフトウェア開発組織での Validation モデルを使った回転率の適用

第四章は，ソフトウェア生産技術によるソフトウェア開発組織全体の定量化事例および成果を示す．従来のソフトウェア開発組織の効率測定として用いられている生産性，すなわち成果量÷コストに加え，組織の俊敏さを計測するメトリクスとして多くの産業分野で広く活用されている回転率をソフトウェア開発においても導入できることを示す．提案するメトリクス，ソフトウェア開発回転率は V&V モデルから派生させた Validation モデルに基づき，ソフトウェア開発組織が機能品質を短期間に開発し，開発するソフトウェアへの要求に対して俊敏に対応できているか否かを表す．ソフトウェア開発回転率は，特定のソフトウェア開発プロセスモデルや開発手法に独立で多様なプロセスモデルに従ったソフトウェア開発プロジェクトを多数持つようなソフトウェア開発組織でも適用可能である．このメトリクスは大規模ソフトウェア開発組織での 9 年以上, 7,000 以上のソフトウェア開発プロジェクトで適用実績がある．本章ではそのうち，約 2800 プロジェクトでの計測結果を報告し，提案したメトリクスの有効性を検証する．

1.　はじめに

同じ機能のソフトウェアを開発するとき，より高品質，より低コスト，より短期間に完成できるソフトウェア開発組織のほうが競争優位である．したがって，どのソフトウェアの開発組織においても，その品質，コストと開発期間を定量的に計測し，改善していくことは他の工業製品事業者と同様重要である．

1970 年代以降，日本の主要なソフトウェアベンダは開発プロセスを標準化し，標準開発プロセスにしたがって, 品質, コスト, 開発期間の計測を行ってきた [69] [59] [3]．しかし，1990 年代以降は，開発プラットフォーム，開発言語，開発拠点といったソフトウェア開発環境を自組織で制御することが困難となり，開発プロセス標準化を前提にした定量的な計測は年々困難になっている．さらに，市場における競争の激化に伴って，ソフトウェアは，常に変化する要求に対していかに俊敏に対応できることが求められるようになり，ソフトウェア開発組織の俊敏さを計測する必要性が増加している．

一方，ソフトウェア開発組織の俊敏さを計測するメトリクスとして，回転率(turnover)またはその逆数であるサイクルタイム(cycle time)が米国を中心に注目されている [7] [70]．回転率は，財務分野では企業の効率を計測する総資本回転率 [71]や，ハードウェア量産品の生産効率を計測する部品在庫回転率 [72]又はその逆数であるサイクルタイム [8]として広く活用されている効率指標であり，このメトリクスをソフトウェア開発に用いて組織の俊敏さ（すなわちアジリティ）を計測しようという提案である．しかし，大規模ソフトウ

ェア開発組織でどのように回転率を計測するのか，例えば，財務指標における総資本や在庫というパラメタはソフトウェア開発組織での計測では何に対応するのかといった回転率の適用方法は明確になっていない．さらに，回転率を実際に大規模ソフトウェア開発組織に適用して，俊敏さの向上を計測した事例も報告されていない．

本章は，大規模ソフトウェア開発組織における組織の俊敏さを計測するメトリクスとして回転率の活用方法を述べる．まず，2 節で大規模ソフトウェア開発組織における計測方法の課題を述べ，3 節では開発プロセスモデルに依存しないソフトウェアの品質確保モデル Validation モデルを提案する．また，このモデルを使ったソフトウェア開発回転率と従来からの指標である品質，コスト，開発期間との関係を明確にする．4 節では，ソフトウェア開発回転率を具体的にどのように大規模ソフトウェア開発組織に適用するかを述べ，約 2,800 の開発プロジェクトに実用し，組織の俊敏さを計測，改善した事例を報告する．最後に，5 節で Validation モデル及びソフトウェア開発回転率の有効性を検証する．

2. 関連分野の動向及び課題

本節では，大規模ソフトウェア開発組織での従来の計測方法を概観し，本章の手法が解決する課題を明確化する．

2.1 日本のソフトウェア工場アプローチとその課題

1970 年代以降，日本の主要なソフトウェアベンダは日本のソフトウェア工場のアプローチ [69] [59] [3] を採用した．すなわちハードウェア製造におけるインダストリアル・エンジニアリングと同様にソフトウェアの開発プロセスを標準化し，標準化された開発プロセスにしたがって，品質，コスト，開発期間の管理を可能にした．このアプローチにおける主要なメトリクスは生産性「生産量÷コスト」 [65] [67] 及び，品質密度「フォールト数÷生産量」 [73] [68]で，これらのメトリクスを向上させることにより，より良い品質のソフトウェアをより効率よく開発する活動を行ってきた．この管理方法は，1970 年代，1980 年代には大きな成果を挙げた．すなわち，安定した開発プラットフォーム，特定の開発言語，トレーニングされた自組織のソフトウェア開発者，画一的な開発プロセスという工場的アプローチの前提条件が満足できる場合には有効な方法であった．しかし，1990 年代以降は，他社が開発したプラットフォーム，多様な開発言語，国内外の開発拠点，プロジェクトの特性に合わせた開発プロセス，といったプロジェクトの多様性が増加し，開発プロセスを標準化することを前提にしたメトリクスの適用は年々困難になってきている．

2.2 反復型の開発プロセスにおける指標の課題

一方，1990 年代以降ソフトウェア市場における競争の激化に伴い，ソフトウェア開発組織は，開発するソフトウェアに対する要求を開発開始時に固定することが困難になり，変化する要求に対して俊敏に対応できることが求められるようになってきた．この課題に対

応して，開発プロセスの分野では，適応的にソフトウェア開発を行う反復型の開発プロセスが多く提案されている．反復型の開発プロセスを採用したソフトウェア開発プロジェクトでも，従来からのメトリクス，生産性「生産量÷コスト」及び，品質密度「フォールト数÷生産量」は，計測可能である．しかし，Basili [53]の言うように，ソフトウェアのメトリクスは，それを計測する組織の目標(Goal)に沿っていることが前提である．すなわち，あるソフトウェア開発組織の目標が，変化する要求に俊敏に対応することの場合，その組織が計測すべきメトリクスは，「変化する要求に俊敏に対応すること」に関連するもので無ければならない．

一般に生産性を高めて効率的に開発すれば，開発期間が短縮され，要求への対応に要する期間も短くなる．しかし，生産性の向上は，必ずしもソフトウェア開発期間の短縮に結び付かない．例えば，図 4- 1 は，同じ機能のソフトウェアを，開発期間，投入人員の異なる二つのプロジェクトで開発するときのガントチャートを示している．開発人員の単価が等しい場合，プロジェクトＡのほうが生産性は良いが，開発期間はプロジェクトＢのほうが短い．財務的な立場で，Denne ら [74]の示すように，生産性の低いプロジェクトＢのほうが早期の収入機会を得て，かつ，新たな機能改善機会を得ることができるため，ビジネス的に優位な場合が多い．例えば，図 4- 1 の例で 2010 年の 1 月に競合他社が同様のソフトウェアを市場に投入した場合, A プロジェクトはビジネス的に失敗する危険性が増加する．したがって，「変化する要求に俊敏に対応すること」という目標を持った組織では，従来から使われている生産性の計測だけでは不十分である．

プロジェクト	見積 KLOC	開発人員	2009 1 4 7 10	2010 1 4 7 10	生産性 KLOC/人月
A	150	6	1/10	'10/9/10	1.25
B	150	25	1/10 7/10		1.00

図 4- 1 生産性と開発期間が矛盾するプロジェクト例

Figure 4-1 A Case Where Productivity Contradicts the Development Period

一方，ソフトウェア開発組織の持つソフトウェア開発プロジェクト全体の平均開発期間が求められれば，「変化する要求に俊敏に対応すること」へのメトリクスになる．例えば，同時期に一つのソフトウェア開発プロジェクトしか運用していない組織であれば開発期間は容易に計測できる．しかし，大規模なソフトウェア開発組織では，常に多くのソフトウェア開発プロジェクトが並行して実行され，さらに，それぞれのプロジェクトは，開始日も終了日も異なり，規模も大きく異なる．このような組織において，ある計測期間での平均開発期間を求めるのは容易ではない．

2.3　仕掛かりを用いた回転率指標およびその課題

2.2 項で示した開発期間計測方法の課題に対して，財務指標などで広く活用されている仕掛かり(inventory)という概念をソフトウェア開発にも適用するアイデアが，Poppendieck [27] [7]（リーン生産手法をソフトウェアに適用），Anderson [70]（制約条件の理論 TOC をソフトウェアに適用）から提案されている．ここで仕掛かりとは，あるプロセスに滞留している量で，財務会計における資産(Asset)や，ハードウェア製造における部品在庫などに相当する．この仕掛かりから，仕掛かかりが単位期間でどの程度入れ換わるか，すなわち，回転率を次のように求めることができる．

待ち行列理論におけるリトルの法則 [23]により，安定した系において，あるプロセスの平均仕掛かり量は，平均入力確率と，プロセスでの平均仕掛かり時間の積に等しい．

$$\text{平均仕掛かり量} = \text{平均入力確率} \times \text{平均仕掛かり時間} \tag{4-1}$$

閉じて安定した系では，プロセスへの平均入力確率はプロセスからの平均出力確率に等しいので，(4-1)式のリトルの法則は下記のように変形できる（ [27] p.122）．

$$\text{平均仕掛かり時間} = \frac{\text{平均仕掛かり量}}{\text{平均出力確率}} \tag{4-2}$$

さらに，(4-2)式の右辺の分母分子に，単位期間(計測期間)を掛ける．分母の平均出力確率は単位期間を掛けることにより，その単位期間での出力量となる．

$$\text{平均仕掛かり時間} = \frac{\text{平均仕掛かり量} \times \text{単位期間}}{\text{単位期間の出力量}} \tag{4-3}$$

(4-3)式をさらに変形すると

$$\text{回転率} = \frac{\text{単位期間}}{\text{平均仕掛かり時間}} = \frac{\text{単位期間の出力量}}{\text{平均仕掛かり量}} \tag{4-4}$$

となり，(4-4)式では両辺とも，単位が相殺される．この量は，回転率(turnover)と呼ばれ，単位は，単位期間あたりの回数で，そのプロセスでの仕掛かりが単位期間中に平均何回入れ換わるかを示している．回転率は会社経営（総資本回転率），工場経営(部品の在庫回転率)，小売業（商品の在庫回転率，店内での顧客回転率）などでプロセスの効率を計測するメトリクスとして広く活用されている単位である．

回転率をソフトウェア開発に当てはめると，あるソフトウェア開発組織の回転率は次の

(4-5)式で表すことができる.

$$\text{ソフトウェア開発組織の回転率} = \frac{\text{期間での成果物の総和}}{\text{期間内の仕掛かりの平均}} \qquad (4\text{-}5)$$

ここで,「期間での成果物の総和」とは計測期間での開発を完了したソフトウェアの総量で,「期間内の仕掛かりの平均」とは,計測期間でどの程度のソフトウェアが仕掛かっているかを示す.すなわち,ソフトウェア開発組織の回転率とは,あるソフトウェア開発組織が,ある単位期間で平均何回プロジェクトが入れ換わっているかを表す.図 4- 2 にソフトウェア開発における回転率の例を示す.

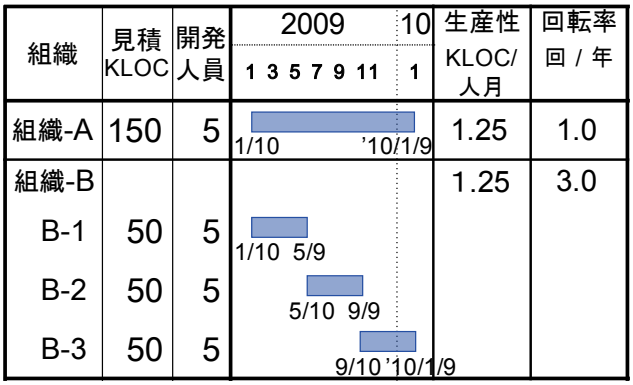

図 4- 2 ソフトウェア開発における回転率の例

Figure 4-2 Turnover Metric of Software Development Projects

同じ開発人員を持つ組織 A と組織 B は,'09/1～'10/1 の 1 年間に同じ量のソフトウェアを開発している.したがって,開発人員の単価が等しい場合,組織 A と組織 B の生産性は等しい.ソフトウェアの仕掛かりを見積もりソース行数だと仮定すると,組織 A の平均仕掛かりは,150KLOC,組織 B は 50KLOC となる(仕掛かりとして見積もりソース行数にする妥当性については,3.2 項で述べる).一方,1 年間の成果量は,組織 A,B ともに 150KLOC となる.したがって,組織 A の回転率は 1 回/年,組織 B の回転率は 3 回/年となり,組織 B のほうが組織 A より 3 倍回転率が高いという結果を得る.

回転率を求めるために必要な量は,計測期間における仕掛かり量と成果量であり,どのような開発プロジェクト編成でどのような開発プロセスを採用しているかとは無関係である.また,図 4- 2 の例は同じ開発量の場合の比較であるが,開発量が違う複数の開発組織で比較することもできる. さらに,ソフトウェアの規模の計測単位(LOC,ファンクションポイント [31]など)が変わっても(4-5)式右辺で,分母と分子の単位が同じであれば,どの単位を使っても回転率では単位が相殺される.したがって,異なった開発方法,異なっ

た単位を使った複数組織での回転率の比較も可能である．すなわち，このメトリクスは，開発プロセスの設定方法やプロジェクトの編成方法には依存せず，さらに，ソフトウェア規模を計測する任意のメトリクスを使用時も適用可能という特長がある．

このように回転率は，組織の敏捷性を計測するために効果的なメトリクスであるが，実際のソフトウェア開発組織に適用する場合，以下の課題がある．

（1）ソフトウェア開発組織の目標との整合性維持

単に，回転が速い，短期間開発というだけでは，ソフトウェア開発組織が求めている，必要な機能品質を短期間に開発し，要求へ俊敏に対応するという目標から外れるリスクがある. 図 4- 2 の例の場合, 組織 B が単に反復型の開発プロセスを採用したというだけでは，組織 A より短期間で開発したとは言えず，要求へ俊敏に対応したかどうかも不明である．このようなリスクを回避するためには，ソフトウェア開発における回転率がどのように，機能や品質に関連しており，また，回転率をソフトウェア開発に適用時，どのような契機でプロセスが開始し，どのような条件でプロセスから抜けるのかといったことを明確にモデリングし，メトリクス化する必要がある．

（2）ソフトウェア開発組織での現実的な適用方法の確立

実際のソフトウェア開発組織で回転率を計測し，改善するためには多様な規模，多様な開発プロセスの複数プロジェクトでの適用方法, 合算方法を決める必要がある．たとえば，仕掛かりをどの時点でどのように計測するのか，どのように複数プロジェクトの結果を合算できるようにするのか等を決める必要がある．さらに，実運用をする際には，計測精度と計測負荷のトレードオフが必要で，どのような計測区間でどのように計測するかといった現実的な適用方法を確立する必要がある．

3. アプローチ

2 節で述べた現状の課題に対し, 組織目標との整合性を図るために本研究で導入したモデルおよびメトリクスを 3.1 項に述べ，3.2 項では実際のソフトウェア開発組織に適用時の単位の設定方法を説明する．

3.1 ソフトウェアの Validation モデルとソフトウェア開発回転率

2.3 項の課題(1)で述べたように，ソフトウェア開発組織が必要とするメトリクスは，単に短期間で開発するだけではなく，必要な機能品質を短期間に確立することである．この課題を解決するために, 本研究は, 2 節で述べた回転率と, ソフトウェアの品質管理分野での, Verification & Validation モデル [75] [76]（以下，V&V モデル）を結合した Validation モデルを提案する．このモデルにより，ソフトウェア開発における「仕掛かり」や「最終成果物」を定義し，開発するソフトウェア全体や，部分としてのソフトウェア構成項目ごと

に開発期間短縮と機能品質確保の両方が適用できることを示す.

１） V&V モデルとは

IEEE のソフトウェア工学用語集 [75]によると verification は「ある工程の成果物がその工程開始時に設定した条件を満足していること」を工程ごとに確認することであり，validation は「開発されたソフトウェアが指定された(specified)要求を満足していること」を通常，最終段階で確認する活動である．verification や validation の対象はソフトウェア全体の場合もあるし，その一部の場合もある．一般には，ソフトウェア開発における中間成果物であるドキュメントやソースコードなどを対象としてレビューやインスペクションといった verification が実施される．その後，主に動作可能なソフトウェアを対象として，元々の要求に合っているか否かを確認する validation が実施される．ソフトウェア開発手法やプロセスモデルにより, verification や validation を実施する順番や実施回数は異なる．しかし，V&V モデルを経て最終成果物になることは，ソフトウェア開発手法やプロセスモデルに独立である．

２） Validation モデル

本研究では，ソフトウェア構成項目(computer software configuration item [75])が計画されてから validation されるまでの状態遷移を V&V モデルに基づきモデル化し，図 4- 3 に示す Validation モデルとして提案する．

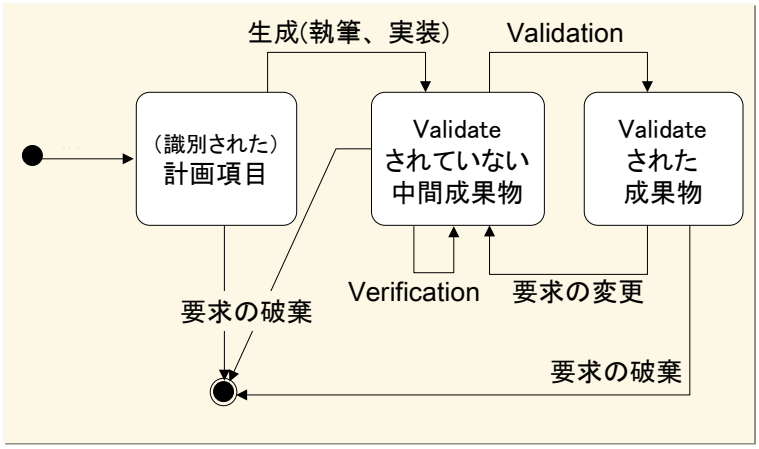

図 4- 3 Validation モデル（ソフトウェア構成項目ごとの状態遷移図）

Figure 4-3 Validation Model (Statechart for a Software configuration Item)

Validation モデルでは，あるソフトウェア構成項目は，計画時に「識別された計画項目の状態」になり，生成開始時に「validate されていない中間成果物」になり，validation により,「validate された成果物の状態」と状態遷移する．このモデルの特徴を以下に述べる．

- 「validate されていない中間成果物」状態にあるソフトウェア構成項目をソフトウェア開発における仕掛かり,「validate された成果物」を最終成果物と明確に定義した.
- verification によって状態は遷移しない. すなわち, 中間成果物の状態のままとした. Validation モデルにおいて, verification は品質確保を行うための手段の一つであり, 最終的な成果物になるためのイベントはあくまでも validation と考えた.
- ソフトウェア開発の場合は，validation の対象となる要求が変更される場合も多い. すでに validate されたソフトウェア構成項目に対して要求変更時には，そのソフトウェア構成項目は中間成果物の状態に戻るという状態遷移も考慮した.

3） ソフトウェア開発回転率

2.3 項(5)式で説明した回転率に Validation モデルを当てはめたメトリクス，ソフトウェア開発回転率は次の(4-6)式で定義される.

$$St = \frac{Sv}{Su} \qquad (4\text{-}6)$$

ここで， St は，あるソフトウェア構成項目のソフトウェア開発回転率．Sv は計測期間で「validate された成果物の状態」になったそのソフトウェア構成項目の総量, Su は,「(作り始めたが) まだ validate されていない中間成果物」状態にあるそのソフトウェア構成項目のその期間の平均を示す.

(4-6)式は 2 節で説明した回転率の式(4-5)と同様であるが，validation，すなわち要求の満足の確認を経ないと仕掛かりの状態を脱しないようにすることにより，「単に回転が速い」といった品質無視の開発方法の濫用を抑止し，「必要な機能品質を短期間に開発し，要求へ俊敏に対応する」という目標に沿った指標にできる．また，V&V モデルに基づいてのソフトウェア構成項目の状態遷移をモデル化したことにより，どのようなソフトウェア構成項目，どのようなソフトウェア開発プロセスを採用したときでも同様に仕掛かり，最終成果物が計測できる.

ここで, 実際のソフトウェア開発のプロセスでどのように「validate されていない中間成果物」が観測されるかを図 4- 4，図 4- 5 を使って説明する．ウォーターフォールモデルで開発している場合，Validate されていない中間成果物の総量は図 4- 4 のようになる．複数の計画項目は最終的にそのプログラムが完成するまで中間成果物として蓄積される．一方, 反復型の開発プロセスで反復ごとに Validation を行ってソフトウェアを開発している場合の Validate されていない中間成果物の総量は図 4- 5 のようになり，単位期間当たりに滞留している仕掛かりはウォーターフォールモデルに比べて小さくなる.

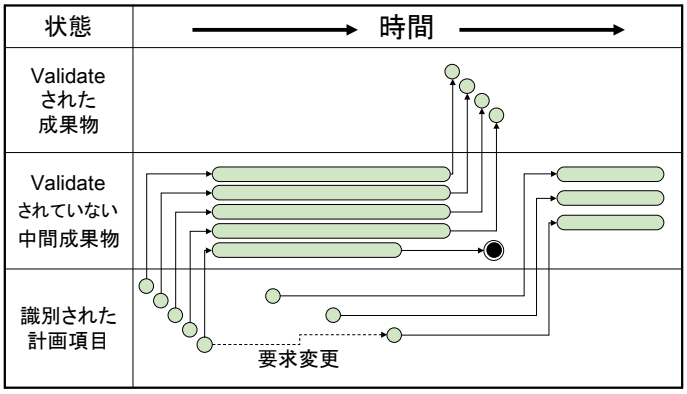

図 4- 4 ウォーターフォールモデルでのソフトウェア構成項目の状態遷移例

Figure 4-3 Typical State Transition of Software Configuration Items Using Waterfall Model

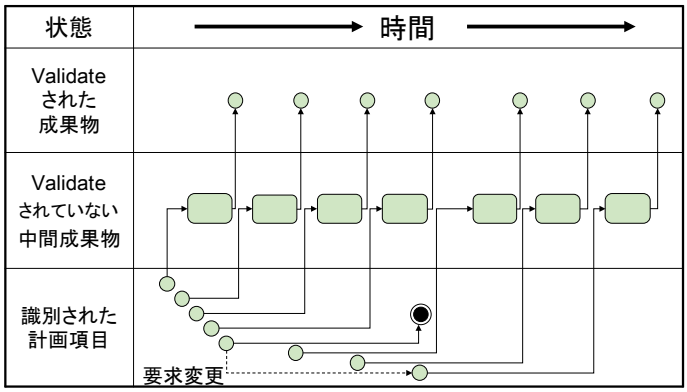

図 4- 5 反復的に Validation するプロセスでのソフトウェア構成項目の状態遷移例

Figure 4-4 Typical State Transition of Software Configuration Items Using Iterative Validation Process

これらの図から明らかなように，ある時点での仕掛かり量は，ウォーターフォールモデルのほうが大きく，結果として，反復型の開発プロセスを採用したほうが，ソフトウェア開発回転率は高くなることが分かる．ここで注意が必要なのは，反復型開発でソフトウェア開発回転率が上がるのは，反復毎に，”Validation を行って”ソフトウェアを開発している場合のみである．反復的な開発プロセスは採用していても，反復毎に，Validation をしないようなプロセスでは，ソフトウェア開発回転率は上がらず，ウォーターフォールモデルと同じになる．アジャイルソフトウェア開発手法では，図 4- 6 のように単に反復型のソフトウェア開発を行うだけでなく，常に動作するソフトウェアを使ってユーザーと Validation を繰り返すことが奨励されている．

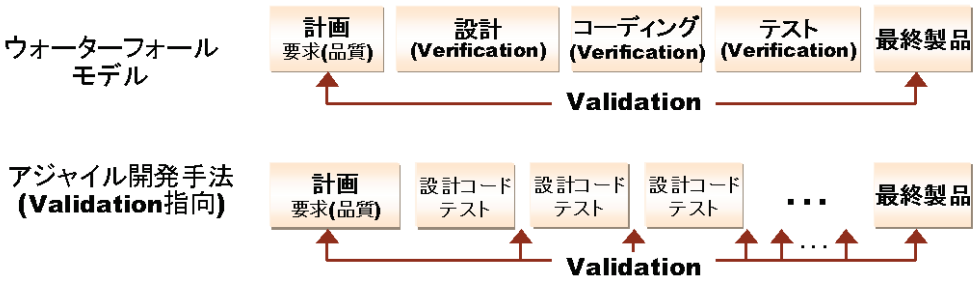

図 4- 6 V&V の適用パターン

Figure 4-6. Typical Application Pattern of V&V

このように，短期間に Validation を繰り返す場合のソフトウェア開発回転率は高いが，単にアジャイルソフトウェア開発手法の他のプラクティスを使って反復開発しているというだけでは，仕掛かり（すなわちリスク）が反復毎に積み重なりソフトウェア開発回転率は上がらない．

3.2　ソフトウェア開発回転率における単位の設定方法

2.3 項の課題(2)で述べたように回転率をソフトウェア開発プロジェクトに適用する場合，その単位期間と仕掛かりをどのように設定，計測するかが課題となる．本項では，まず回転を計測するための量を何にするかを述べ，次に期間や，平均の求め方についての本研究のアプローチを説明する．

（1）　仕掛かり量，成果量の計測方法

ハードウェアの量産品のように，成果物としてのソフトウェアが全て同じであれば，仕掛かり量，成果量の計測は，ある時点で実行しているソフトウェア開発プロジェクト数を計測すれば良い．しかし，ソフトウェアの場合にはプロジェクトによって成果物は異なるので，何らかの重みづけが必要である．理想的には，開発するソフトウェアが出荷後どのような価値を生み出し，どの程度の収益が得られるかという観点で重み付けをすることが考えられる．Denne ら [74]の示す IFM はこのような計測を可能にする方法論であるが，これを多数のプロジェクトに対して実施するのは，計測負荷が高い．実施するソフトウェア開発組織の開発プロセス成熟度が高いことを前提に，現実的には，開発するソフトウェアの見積もり規模によって重みづけすることを提案する．理由は，見積もり規模は回転率の計測とは関係なく，ほとんどのソフトウェア開発でデータとして採取しているため，計測負荷が低くなること，また，IPA SEC の発行するソフトウェア開発データ白書 19) pp.247-249 によれば，規模見積もりを行っている 267 プロジェクトのうち，実績との誤差が 20%以内のプロジェクトは 201 プロジェクト約 75%あること，見積もり規模と実績の規模の中央値はほとんど同じであることである．

なお，「仕掛かり」という語感から，ソフトウェア開発プロジェクトが実行中に，仕掛か

り量を漸次的に増加するように計測する方法は誤りである．ソフトウェア開発プロジェクトの回転率における仕掛かりとはあくまで仕掛かっているプロジェクトの成果物全体の価値を（代用して）計測するものであり，プロジェクトの中間段階での出来具合で変化するものではない．Validation モデルが示す通りプロジェクトが計画されてから，成果物全体が Validation されるまでの間は全てが仕掛かりとして計測する必要がある．

（2） 単位期間，仕掛かり平均の求め方

他の産業分野での単位期間の設定は計測する対象によって異なり，総資本回転率の数ヶ月～1 年から，顧客回転率のように数時間～1 日単位のものまである．単位期間は，仕掛かりがプロセスに滞留している期間の長短によって決められるものである．したがって，ソフトウェア開発プロジェクトに回転率を適用する場合，まず考慮すべきなのは，平均的なソフトウェア開発プロジェクトの開発期間である．IPA SEC のデータによれば，ソフトウェア開発プロジェクトの開発期間のモードは２～４ヶ月で，中央値は 6.8 ヶ月である．このため，３ヶ月～1 年程度の単位期間で計測するのが良いと考える．

仕掛かり平均の他産業分野での一般的な算出方法は，単位期間の開始時と終了時の仕掛かり量の平均を用いる方法である．ソフトウェア開発プロジェクトでは，実行するソフトウェア開発組織の成熟度が高い場合，各プロジェクトの見積もり量，開始日，完了(予定)日は管理されている．したがって，他産業分野のような開始と終了時の平均といった粗い計測方法ではなく，細かな管理を行うことが可能と考える．また，ソフトウェア開発プロジェクトの場合，月単位に，月の初めや月の終わりにプロジェクトの開始や終了を設定する場合が多く，１ヶ月未満の間隔で仕掛かりを計測すると誤差が多くなる．したがって，ソフトウェア開発組織全体の仕掛かりを計測する場合，１ヶ月ごとに仕掛かりを計測し，単位期間で平均をとる方法が良いと考える．

4. 大規模ソフトウェア開発組織での適用実例

3 節で示した Validation モデルにしたがったソフトウェア開発回転率の実用性と有効性を検証するために，大規模ソフトウェア開発組織に適用した事例を報告する．検証する項目は，次の項目である．

- ソフトウェアの開発スピードアップという組織目標をソフトウェア回転率で表現できるか否か
- ソフトウェア開発回転率と従来の指標ソフトウェア生産性との差異があるか
- 単なる反復ではなく，Validation モデルを導入したことによって開発スピードがアップしても品質への影響が少ないこと
- ソフトウェア開発回転率の測定負荷は，大きくないことの検証

4.1　組織概要及び適用対象

日立ソフトウェア事業部（以下，ソフトウェア事業部）は，約 4,000 人のソフトウェア開発者を擁し，パッケージ型のソフトウェアを開発する組織である．1969 年に設立後，1970 年代から 2.1 項で述べたソフトウェア工場アプローチにより，ソフトウェアの生産性，信頼性の両面で定量的な目標を決めて管理推進している．1990 年代前半までは，開発期間や中間プロセスを意識しない，「成果量÷コスト」，「フォールト数÷成果量」といったメトリクスを活用していた．しかし，1990 年代以降，市場での競争激化に従い，「効率向上」「信頼性向上」に加えて「開発スピードアップ」も組織目標としている．この組織目標に対応して，1990 年代後半以降，生産管理，品質管理の両面で開発期間や中間プロセスを意識した定量的な目標を立てて推進している（表 4- 1）．

表 4- 1 生産技術，品質管理の重点項目

Table4-1　Priority Targets of Software Industrial Engineering and Quality Management

	生産技術	品質管理
1970 年代 後半～	生産性， 納期遵守， etc.	最終工程でのフォール ト摘出数， フィールドでの障害数， etc.
1990 年代 後半～	（上記に加え） アジリティ， 多様な開発形態に対す る生産性の正規化	（上記に加え） 各フォールトの摘出すべ き上流工程の特定による 定量的評価

1999 年 10 月より事業部で実行される全てのソフトウェア開発プロジェクトを対象にソフトウェア開発回転率の計測を開始し，2009 年 3 月現在，7000 以上のプロジェクトでのデータが蓄積されている．

4.2　Validation モデルの具体的適用方法

本項は，3 節で示したアプローチに基づき，ソフトウェア事業部が具体的に Validation モデルをどのように適用し，多数のソフトウェア開発プロジェクトのソフトウェア開発回転率を計測しているかを説明する．

計測間隔

当該組織では１ヶ月ごとに仕掛かりおよび成果量を計測し，６ヶ月毎にソフトウェア開発回転率を計測している．ソフトウェア開発回転率計測に必要なデータは，それまでの生産性管理で使用していたデータをそのまま使用し，データ集計時に新たな集計ルーチンを加えることで対応した．

平均仕掛かり量　Su

1ヶ月ごとに，その時点で開発中のソフトウェアプロジェクトの開発量の総和を，その時点での組織の仕掛かりとし，ソフトウェア開発回転率を計測する期間，6ヶ月間での仕掛かりの平均を，平均仕掛かり量 Su とした．ここでの規模は開発するソフトウェアの実際にコーディングが完了しているソース規模ではなく，プロジェクト全体の見積もりソース規模(ソースコード行数)を各月仕掛かりとして計測している．ソフトウェア開発プロジェクト回転率を計測する場合，3.2 項(1)で述べたとおり，そのプロジェクトが計画されてから validation されるまでは，そのプロジェクトの計画全体を仕掛かりとして計測している．

成果量　Sv

全てのソフトウェア開発プロジェクトは，その終了時に独立した品質保証部署によって validation が実施される．ソフトウェア開発回転率を計測する期間の6ヶ月間で validation が完了したソフトウェア開発プロジェクトの開発量（実際に開発したソースコードの量）の総和を，成果量 Sv とした．この単位は平均仕掛かり量 Su と同じくソースコード行数である．

ソフトウェア開発回転率　St

成果量 Sv を平均仕掛かり量 Su で割ることにより，計測する期間6ヶ月間のソフトウェア開発回転率 St を計測している．上記の方式を使った具体的なソフトウェア開発回転率の算出例は，付録 A を参照のこと．

なお，2.3 項で述べたとおり，Validation モデルおよびソフトウェア開発回転率を適用する場合，対象は閉じた安定した系である必要がある．例えば，計画されたが中途で中止になるようなプロジェクトが多数あったり，開発量が計測区間内で大きく偏りがあったりするような場合はソフトウェア開発回転率の精度が落ちる．ソフトウェア事業部では，途中で中止するようなプロジェクトの確率は 5%未満で閉じた系だと見なすことが可能である．また，計測対象は安定した数の開発者を擁する組織であるため，6ヶ月という計測間隔においては系への入力，系からの出力は大きく変わらない．従って，閉じて，安定した系として扱うことが可能である．

4.3　Validation モデルの適用対象

本章では，成長市場での開発スピード向上を目指した 4 年間を対象に開発期間短縮を目標とした施策を推進したグループと，推進しなかった他のグループへの適用結果を報告する．表 4- 2 に両グループの概要，特徴と主な開発施策をまとめた．両グループのうち，グループ B は計測開始時点で開発期間短縮を目標とした．この組織の多くの開発プロジェクトは，増分反復型開発（インクリメンタルモデル） [77]の適用，プロトタイピング，コンカレント QA（品質保証部署が行う validation を開発プロセスの最終段階だけでなく，開発

の中途段階で実施する施策）といった施策を計測期間に採用し，開発期間の短縮を組織目標とした．

表 4- 2 計測したグループとその特徴

Table4-1　Outline of Groups

	グループＡ	グループＢ
プロジェクト数	1,649	1,185
平均プロジェクト規模	13.7 KLOC	26.8 KLOC
メトリクス (両グループ共通)	ソフトウェア構成項目：見積ソース行数 メトリクス：6 ヶ月のソフトウェア開発回転率	
計測期間	4 年間（同時期）	
主な開発プロセス モデル	ウォーターフォール	ウォーターフォール， 増分反復型開発
対象市場	成熟市場	成長市場
採用した主なソフトウェア 開発技術	テスト自動化 母体解析 形式的検証試行	プロトタイピング コンカレントＱＡ 部品活用

4.4　ソフトウェア開発回転率及び生産性の推移

ソフトウェア開発回転率の期（6 ヶ月）ごとの比較結果を図 4- 7 に示す．また，同時期の「成果量÷コスト」の生産性の比較結果を図 4- 8 に示す．両方の図とも，横軸は，計測期間(期単位)，縦軸はグループ B の最初の期の結果を 1 と仮定したときの相対値である．

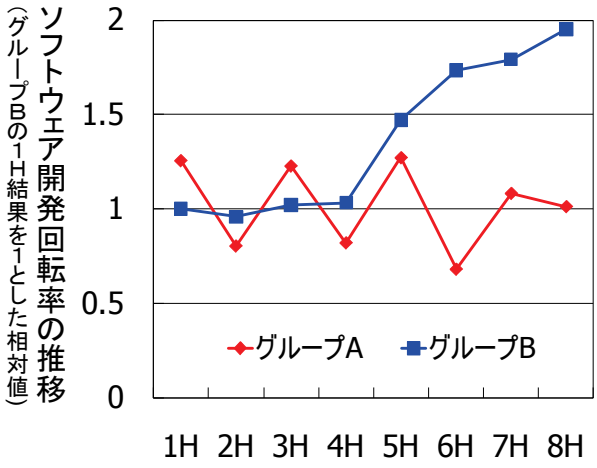

図 4- 7 ソフトウェア開発回転率の推移比較

Figure4-4 Changes in Software Development Turnover of the Two Groups

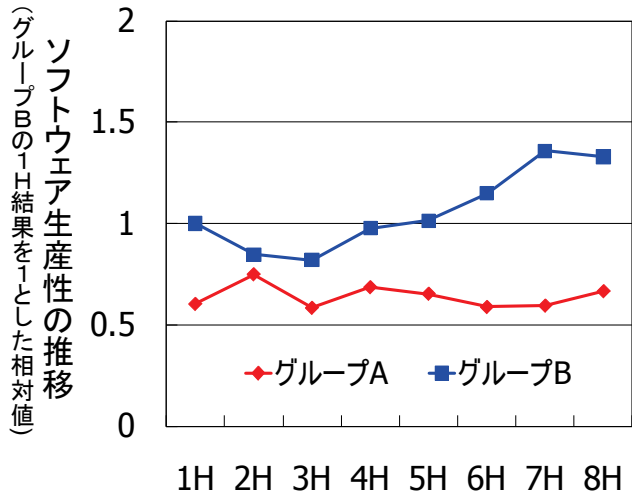

図 4- 8 生産性の推移比較

Figure4-8 Changes in Productivity of the Two Groups

図 4- 7 の結果により，開発期間短縮向上施策を実施したグループ B は 4 年間で約 2 倍のソフトウェア開発回転率の向上を計測したが，開発期間短縮向上施策を実施しなかったグループ A はほぼ変化が見られなかった．図 4- 8 の生産性の結果と比較すると，ソフトウェア開発回転率の推移と生産性の推移は，必ずしも同じ傾向を示さないことが分かる．グループ A のソフトウェア開発回転率が期ごとにアップダウンしているのに対して，生産性は大きな変化が見られない．また，グループ B の生産性が 1 低下した 2H から 3H，7H から 8H でも，ソフトウェア開発回転率は向上している．

グループ B のソフトウェア開発期間（ソフトウェア開発回転率の逆数）の分布を図 4- 9 に示す．上の図は改善前の期(最初の 6 ヶ月)，下の図は最後の期(最後の 6 ヶ月）のソフトウェア開発期間の分布である．また，図 4- 10 は，図 4- 9 の分布の累積グラフである．図 4- 9，図 4- 10 からも明らかな通り，改善前に比べて開発期間は平均，モードともに半減，メディアンは 30%短縮した．ただ，改善前，改善後ともに，標準偏差が 100 日以上と大きな分散がある．また，改善前，改善後とも標準偏差にほとんど変化が無く，改善前の標準偏差を 1 とした場合，改善後の標準偏差は 0.97 であった．

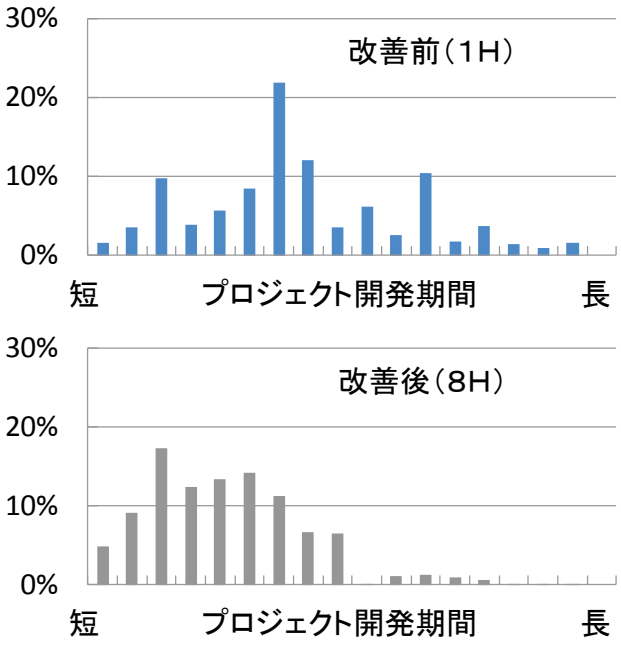

図 4- 9 プロジェクト開発期間の分布

Figure4-9 Distribution of Development Periods

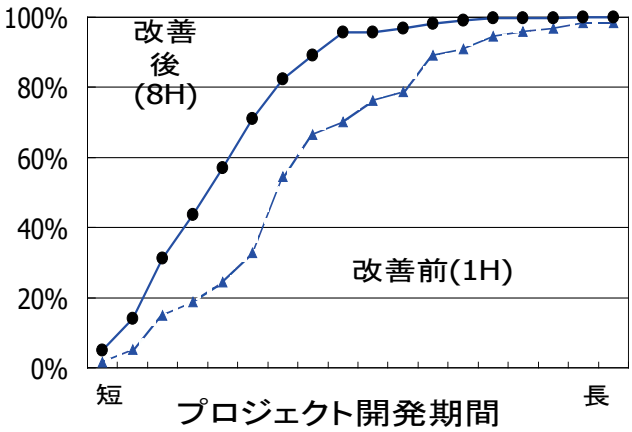

図 4- 10 プロジェクト開発期間の累積分布

Figure 4-7. Cumulative Distribution of Development Periods

5. 考察

5.1 Validation モデルの特長，考慮事項

（1） Validation モデルの特長

Validation モデルは，ソフトウェア構成項目がどのように validate されるかに着目してモデル化した．このモデルは開発プロセスモデルとは独立なため，現状ウォーターフォールモデルに近いプロセスで多数のソフトウェア開発プロジェクトを推進している組織が，

漸次，他のプロセスモデルに切り替えたり，ウォーターフォールモデルのまま，開発人員を増加させて開発期間を短縮したりするような施策をとった場合でも，その効果を組織全体に適用することが可能である．

また，Validation モデルは，ソフトウェアの品質確保方法 validation に焦点を当てたことにより，単に開発期間が短いというだけでなく，最終的にそのソフトウェアに対する要求が早急に充足できたことを計測可能である．また，組織全体で使用できるだけでなく，個別のソフトウェア構成項目の効率改善目標を持ったソフトウェア開発プロジェクトでも計測可能である． すなわち，3.1 項に示したソフトウェア開発回転率だけでなく，最終的にvalidateされるすべてのソフトウェア構成項目(機能要件やソフトウェアフォールト等)でこのモデルの適用が可能である．例えば，ソフトウェアフォールトに対して識別から修正，確認まで管理しているソフトウェア開発組織であれば，新たな管理システムを導入しなくてもその組織がどの程度迅速にフォールトを解決しているかを回転率で定量的に計測することが可能である．

（2） Validation モデル適用時の考慮事項

ソフトウェア構成項目を選ぶ場合，Validation モデルで対象となる項目は，「validate された成果物」という状態を取りうるものである．したがって，内部に組み込まれたライブラリのように外部機能を持たないソフトウェアは本モデルの対象外となる．一方，動作可能なソフトウェアはなくても，厳密に定義された外部要件定義書のように，その要件のステークホルダが validate できる場合がある．このような項目は本モデルの対象となる．また，選択されたソフトウェア構成項目は構成管理されている必要がある．構成管理されていない場合，その項目の識別，状態の変更を構成管理できるようにする必要がある．

5.2 適用結果の考察

（1） ソフトウェア開発スピードアップの計測指標

図 4- 7 に示したように，開発スピードアップを目指したグループ B のソフトウェア開発回転率の値が伸びている．個々のプロジェクト単位ではなく，グループ B という組織レベルでの開発スピードアップをソフトウェア開発回転率で計測できるようになったと考える．

（2） ソフトウェア開発回転率と生産性の比較

図 4- 7，図 4- 8 の比較から明らかなように，ソフトウェア開発回転率の推移と生産性の推移は異なった結果を得た．グループ B の施策のうちコンカレントＱＡのように開発部署と品質保証部署が並行に作業を実行させるような施策は，開発期間を減らすことはできるが，生産性は上がらない場合が多い．グループ B の組織目標は開発期間短縮であり，その目標をソフトウェア開発回転率により適切に定量化できたと評価する．

ただ，グループ B の改善後でも，ソフトウェア開発期間は 100 日以上の大きな分散を持

っている．したがって，生産性管理で行っているように，正規分布などの統計分布に当てはめて開発期間を管理することは困難であることが分かった．これは，開発期間のほうが生産性よりもプロジェクトの特性に大きく依存することに由来する．組織全体としての開発期間を縮めたとしても，依然として長期間のプロジェクトが必要な場合はあり，開発期間の分散を組織全体で管理する意義は大きく無いと考える．

（3）　品質への影響

生産性向上や，開発期間の短縮は，ソフトウェアの最終品質を損ねる危険性がある．同時期のグループ B のフィールドでの品質状況の推移を図 4- 11 に示す．

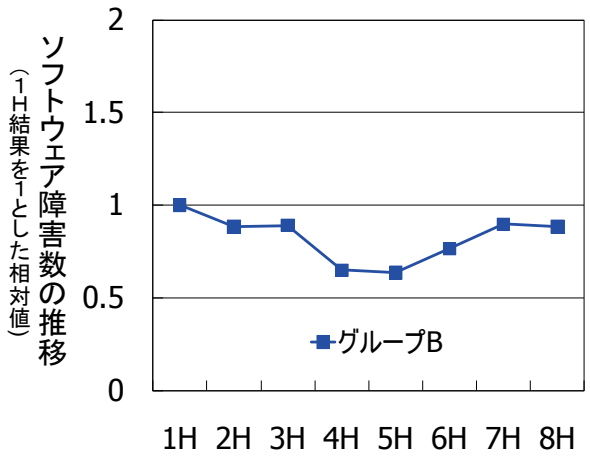

図 4- 11 グループ B のフィールドでの障害数推移

Figure 4-8. Changes in Field Failures of Group B

縦軸の品質は，その期での社外でのソフトウェア障害の発生数を，その期の開発規模で割った．また，最初の期の値を 1 とした相対値でプロットしている．この図では，顕著なフィールド品質の向上は見られないが，ソフトウェア開発回転率の向上によって，ソフトウェアの品質が損なわれていないことが分かった．本結果を以って一般的に開発期間短縮施策が，ソフトウェアの最終品質を損ねていないと結論付けることは困難である．しかし，単なる開発期間短縮ではなくソフトウェアの validation も含めて考える Validation モデルを使ったことによってソフトウェア品質が確保できる事例があることは示せたと考える．

（4）　ソフトウェア開発回転率計測の運用負荷

本節で実例報告したプロジェクト単位のソフトウェア開発回転率の計測データは，すでに生産性計測用に採取していたデータをそのまま流用し，集計ルーチンを追加することだけで計測可能であった．また，対応者の増員も無く，ソフトウェア開発回転率を計測した

ことによるランニングコストの増加は無かった．すでに，組織的なデータ採取の仕掛けが整っているソフトウェア開発組織であれば，データ集計の一時コスト増のみでソフトウェア開発回転率が計測可能であることを実証した．

（5） ソフトウェア開発回転率計測で摘出可能な課題

ソフトウェア開発回転率適用開始時には想定していなかった，ソフトウェア開発回転率の効果があった．すなわち，ソフトウェア開発回転率の計測により，生産性の計測だけでは，顕在化しないソフトウェア開発プロジェクトの課題が摘出できることが分かった．図 4- 7 のグループ A は期ごとにソフトウェア開発回転率がアップダウンしている．この理由は，まず，ウォーターフォールモデルを採用した大規模プロジェクトの多くは，1 年単位でプロジェクトを組んで，特定の月に完了する傾向が見られること，さらに，多くの中小規模プロジェクトが，大規模プロジェクトと同期する必然性が無いにも関わらずスケジュール，特に完了日を合わせようという傾向が見られ，完了日近辺での工数（コスト） をかけないままアイドリングされる傾向があることが分かった．この結果，アイドリングしているプロジェクトにはコストはかからないので，生産性の悪化は計測されないが，ソフトウェア開発回転率で計測することにより，仕掛かりの季節変動という形で顕在化した． ソフトウェアの仕掛かりの季節変動は，顧客事情が原因の場合も多く，必ずしも問題とは言えないが，季節変動が顕在化することによりソフトウェア開発組織側の課題，例えば，人員のスケジューリングの問題などが明確になる場合もある．すなわち，ソフトウェア開発回転率は，単に開発期間の短縮を計測するだけでなく，ソフトウェア開発組織の持っている，各種リソースを効率的に使用しているかどうかをチェック目的にも使えることが分かった．

6. おわりに

本章は，大規模ソフトウェア開発組織でのソフトウェア開発期間計測方法の課題を示し，システム工学でのリトルの法則および，経営工学で活用されている回転率の考えとソフトウェア工学での V&V モデルを組み合わせた Validation モデルを提案した．このモデルにより，ソフトウェアの仕掛かり期間を中間成果物が生成されてから validate されるまでの期間と定義し，多様なソフトウェア開発プロセスに独立なソフトウェア開発回転率メトリクスを示した．また，大規模組織への適用を通じて，ソフトウェア開発回転率の組織メトリクスとしての有効性，生産性との差異，スピードアップと品質の両立および適用の容易性を実証した．また，ソフトウェア開発回転率が，2 章のソフトウェア生産技術のモデルでのソフトウェア生産技術の対象をどの程度効率的に活用しているかという観点でも使用可能なことが分かった．

今後は，提案したモデル，メトリクスを継続して適用，評価することにより，改善していく．特に，今回の事例では成熟度の高いソフトウェア開発組織でのソフトウェア開発プロジェクトの回転率の計測実例を報告したが，成熟度の低く，安定した系を示さないよう

な組織での適用方法や，ソフトウェア開発プロジェクトの中の Validation 可能なソフトウェア構成項目への回転率の適用方法を検討する．また，このモデルを使い，ソフトウェア開発プロセスを最適化する手法も検討する予定である．

四章の付録１　ソフトウェア開発回転率の計測例

4.2 項に示したソフトウェア回転率の計算例を示す. 表 4- 3 のように 4 つのソフトウェア開発プロジェクト A～D を持つ組織の 2009/4～2009/9(網掛け部分)のソフトウェア開発回転率を求める．

表 4- 3 プロジェクト単位でのソフトウェア開発回転率計測例

Table4-3 Measurement of Turnover of Software Development Projects

年/月	各プロジェクト仕掛かり(KLOC)				合計(KLOC)		
	A	B	C	D	仕掛かり	成果物	
09/03	50				50		
09/04	50				50		
09/05	50	30			80		
09/06	50	30	105		185		
09/07	50	リリース	105	50	205	30	
09/08	リリース		105	50	155	50	
09/09			105	リリース	105	50	
09/10			105		105		
09/11			105		105		
09/12			105		105		

計測期間

まず，計測期間 6 ヶ月の平均仕掛かり量 Su，成果量 Sv は下式になる．

$$Su = (50+80+185+205+155+105) / 6 = 130 \quad KLOC \tag{4-7}$$

$$Sv = 30+50+50 = 130 \quad KLOC \tag{4-8}$$

したがって，この例でのソフトウェア開発回転率 St は

$$St = Sv / Su = 130 / 130 = 1 \quad 回/6\ ヶ月 \tag{4-9}$$

となり，6 ヶ月に平均 1 回プロジェクトが入れ替わることが算出できる．

四章の付録２　ソフトウェア構成項目のソフトウェア開発回転率の計測例

本付録では，ソフトウェア開発組織や，その中の一つのソフトウェア開発プロジェクトにおいて，ソフトウェア構成項目のソフトウェア開発回転率を測定する例を示す．ここで説明するのは，あるソフトウェア開発プロジェクトでのソフトウェアフォールト(software

fault)管理でソフトウェア開発回転率を使う，すなわち，いかに識別されたフォールトを迅速に対策確認しているかを計測する例である．

モデル

図 4- 3 の Validation モデルをフォールト用に書き換えたのが図 4- 12 である．ソフトウェア開発において，発生した故障(failure)が識別され，その原因がソフトウェアのフォールトと判明した場合，そのフォールトは識別され未確認の状態となる．その後，フォールトの修正がなされ，最終的に修正済みのソフトウェアが動作確認(すなわち Validation)される．図 4- 12 に示すように，Validation モデルでは，フォールトの修正やレビューでは状態は遷移せず，フォールトの識別から確認までの間は未確認ソフトウェアフォールトとして仕掛かりとする．

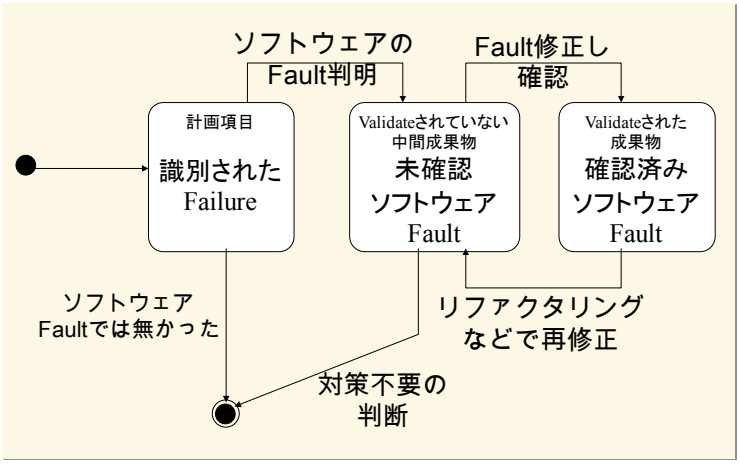

図 4- 12 ソフトウェア Fault に適用した Validation モデル

Figure 4-12. Validation Model Application for a Software Fault

平均仕掛かり量　Su

未確認フォールトの総数を仕掛りとする．したがって，計測する期間の未確認フォールト総数の平均が平均仕掛かり量 Su となる．フォールトのように，一つ一つに重要度の差があるようなソフトウェア構成項目の場合，重み付けをすることも可能である． 1 回/週よりも多くの頻度で未確認フォールト数を計測すると良い.

成果量　Sv

計測する期間中に，（フォールト対策済みのソフトウェアで）確認が終わったフォールト数を，成果量とする．仕掛かりの計測でフォールトの重要度にしたがって重みをつけた場合は，成果量のところでも同じ重みを適用する必要がある.

ソフトウェア開発回転率　St

ある期間での，対策確認済みフォールト数 Sv を平均未解決フォールト数 Su で割ること

により，計測する期間でのフォールトソフトウェア開発回転率 St を求めることができる．

計測例

日本の大規模ソフトウェア開発組織では，ソフトウェア開発のテストプロセスで，テスト実施時間と発見・除去されたソフトウェアフォールト数の関係を図 4- 13 のような S 字形成長曲線により記述し，管理している [78]．このグラフの主目的はソフトウェア内の潜在フォールト数推定やテスト項目消化確認といったテスト進捗度管理である．このグラフをそのまま使い，フォールトに対するソフトウェア開発回転率を計測することできる．すなわち図 4- 13 の破線で囲まれた計測区間で，対策確認をしたフォールト数を成果量(Sv)と見なし，計測区間における未確認フォールト数の平均を仕掛かり(Su)と見なすことによりフォールトのソフトウェア開発回転率(St = Sv/Su)を求めることができる．

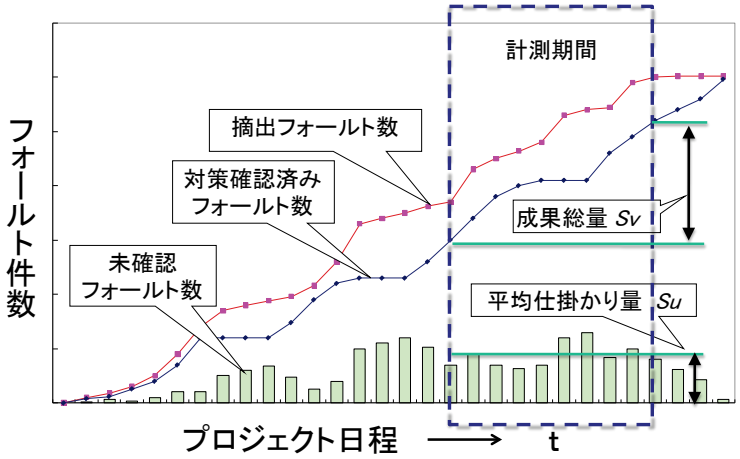

図 4- 13 S 字型成長曲線を使ったフォールト数のソフトウェア開発回転率測定

Figure 4-13. Measurement of Turnover of Software Faults using S-curve Reliability Growth

ソフトウェア開発において管理される要求項目，変更要求，懸案，ソース，ドキュメントといったソフトウェア構成項目のうち，要求項目，変更要求などの validation 可能なソフトウェア構成項目は，本節で示したフォールトと同様な方法を用いて，ソフトウェア開発回転率で管理が可能である．また，本項での例は一つのプロジェクトでの計測例であるが，複数のプロジェクトを持つ組織が個別の validation 可能なソフトウェア構成項目のソフトウェア開発回転率を計測することも可能である．

第五章　コラボレーション活性化と企業活動の適正化を両立させる企業情報システムモデル

第五章は，ソフトウェア開発組織での知識の継続的改善方法および事例を述べる．多数のソフトウェア開発プロジェクトを常時運用する組織は，複数プロジェクトを跨り，組織全体で知識を積極的に共有したほうが良い．電子メール，Web ベースの電子掲示板等の情報ネットワークを基盤とした電子的な手段が企業内での情報交換の手段として主流になっている．近年，さらに電子的なコラボレーションを可能にする手段，Wiki，Blog，SNS 等のサービスを企業内業務に適用したいという要求が高まってきた．一方，企業は個人情報保護，セキュリティ確保や各種規制に対応する目的で電子的なコラボレーション手段を抑制することも求められている．特にソフトウェア開発組織は自組織以外の知的財産権を管理している場合が多い．本章では，この相反する要求を両立させることを目的とし，企業内のコラボレーションシステムの構成，活用される事業の段階ごとにコラボレーションに対する要件を明確にし，それらの要件を満足させる管理モデル，ＯＦＦコラボレーションモデル（Open, Flexible, and Formal collaboration model)を提案する．提案したモデルを大規模組織で適用し，企業内のコラボレーションシステムに求められる要件を満たすことを確認する．

1.　はじめに

多くの企業は事業遂行のためにインターネット由来の情報技術を企業内ネットワークで活用している．電子メールベースのオフィス内の連絡，Web ベースの情報伝達，インターネットや携帯電話を使用した営業活動，等の情報システムを利用したビジネスプロセスが日常となった．特にソフトウェア開発組織では，生産するソフトウェア自体が情報であり，企業内の情報ネットワークは，ソフトウェア開発プロセスの最も基盤となる開発環境であり，ソフトウェアそのものだけでなく，ソフトウェア開発に必要な知識も情報ネットワークを介して交換し，情報ネットワーク上のデータベースに蓄えられている．

昨今，情報ネットワークの相互接続性と利便性を活用し，より事業の成果を上げかつ事業を効率化しようという動きがある．先進的な企業では，Wiki のように多数の人間が共同して一つのコンテンツを作成するツールや，社内ブログ，社内 SNS(Social Networking Service）のように個人が不特定多数の人間に発信したり，社内での人のつながりを新たに作ったりするようなツールを社内の業務にも適用しようとしている [79]．しかし，電子メールや Web ベースの情報発信が比較的スムーズに企業内にも受け入れられたのに対して，Wiki，SNS 等のコラボレーションを促進するツールを活用する企業は少なく，また，活用できる分野も限定的な状況である [80].

本章では，コラボレーションツールを使った情報システムをコラボレーション基盤と定義し，企業内のコラボレーション基盤を情報システムの運用という観点で現状の課題を明確化し，その活性化を実現できる企業情報システムモデルを提案し，そのモデルの有効性を確認するために，大規模組織へ適用し評価を行った．

2. 現状の課題

電子メール，Wiki，Blog，SNS などのコラボレーションツールは，不注意な使用による情報漏えいなどのコンプライアンスリスクを無視できない．個人情報保護，顧客情報の保全，適正な企業活動などの今日的な課題に対応するためには，リスク分析に基づいたシステムセキュリティ設計を行い [81]，コラボレーション基盤の運用において適切な制御が必要である．

すでに企業で活用中の電子メールや Web ベースのツールの運用では，ファイヤウォール，ゲートウェイ，プロクシなどの標準プロトコルに従ったツールがあり，これらのツールを活用した社内コラボレーションの活性化とコンプライアンスリスク低減の両立を目指している．例えば，山井ら [82]は，メールゲートウェイを活用したセキュリティと大規模組織の管理容易化手法を提案している．

しかし，Wiki，SNS 等の比較的新しいコラボレーションツールでは，それらツールの本質が，Wiki の場合「だれもが編集できるコンテンツ [83]」であり，SNS の場合は，「人と人とのつながり」のため，ツールで自動的に適切に制御することは困難でコラボレーション基盤管理者に人的な負荷がかかる [80]．

この問題を解決するため，コラボレーション基盤管理者の負荷を軽減することを目的とした，コラボレーションツール側の監視機能や運用機能の強化 [84]や，社内でのコラボレーション基盤自体の使用範囲を運用側で管理できる範囲に狭める [80]などの対策が考えられている．

コラボレーションツールの監視機能，運用機能強化は，セキュリティ強化，コラボレーションツール運用時の管理負荷低減という観点では効果がある．しかし，この対策はコラボレーション基盤のもともとの目的である相互接続性や利便性を制限するという問題とともに，企業内利用者を監視するツールとなり，フレキシブルな協調活動をむしろ阻害する危険性もある．

コラボレーション基盤運用側の管理負荷が高いため適用範囲を限定するという対策は，限定する理由が経営判断や利用者側の要求であれば理解できるが，運用側の負荷という一方的な理由で制限された場合，組織全体で適切かと言う観点で問題になる可能性がある．

企業内でのコラボレーション基盤を情報システム運用の観点でみた現状の課題は次の三点にまとめられる．

（1） コラボレーション基盤の管理不足による情報漏えい，セキュリティ事故等の企業

リスク

（2） コラボレーション基盤を管理過剰にすることによるフレキシブルな協調活動の阻害

（3） コラボレーション基盤運用時の運用者の管理負荷増大に対する対応

表 5- 1 コラボレーション基盤の課題とリスク例

Table 5-1 Collaboration Infrastructure Issues and Risk Examples

現状の課題	想定されるリスク例
コラボレーション基盤の管理不足，統制不足	ネットトラブル（あらし，炎上，祭り等）
	著作権，プライバシーの侵害
	情報漏えい
	クラック，災害
	運用スキル不足によるサービス低下
	同種システムの乱立
	業界規格，規制違反
コラボレーション基盤の管理過剰	自由な発想が阻害
	自由な交流が阻害
	開発コストの増大
運用者の管理負荷増大に対する対応	運用コストの増大
	システム改善頻度が低下
	厳格管理でサービス低下
	エンドユーザの操作性低下

3. コラボレーション基盤管理の考え方

本節では企業内でのコラボレーション基盤で適切な管理を検討するにあたり，前節で述べた課題の整理と，解決の方向性を述べる．まず，現状の課題から想定されるリスクを抽出し表 5- 1 に示した．表 5- 1 に示した想定されるリスク例は，情報セキュリティマネジメントシステム規格 [85]のセキュリティに対する脅威の例にコラボレーションに対するリスク例を追加した.

抽出されたリスク例は，対策案が矛盾してしまうリスクが多い．すなわち，情報漏えい対策，クラック対策といったセキュリティを強化すれば，運用者の管理負荷は増大し，管理負荷を減らせば，エンドユーザの操作性は下がり，管理を厳しくすれば自由な発想は阻害される，というようにコラボレーション基盤全体でリスクの対策案を完全に両立させることは困難である.

この企業の開放性とセキュリティの問題に対し，梅田 [86]は「トレードオフという概念が重要」で,「かけたコストに対して得られる効果，あるいはリスクを冷静に評価し，判断する姿勢が重要」と述べている．しかし，筆者は開放性とセキュリティを安易にトレードオフするのではなく，コラボレーション基盤の管理する対象又は，管理する場面を適切に分割し，分割された部分ごとに適切な対策を考えることでセキュリティと開放性の両立が

可能であると考えた.

例えば，企業でのコラボレーションツールを利用した情報システムの構成やその活用方法をみると，本質的に人間が制御しなければならない部分もあるが，ソフトウェアによる機械的な監視や機能制限が有効な部分もある．また，組織全体的で横断的に管理しなければならない業務もあれば，特定部署，プロジェクトやコミュニティ等の管理者に管理を委譲しても問題無い業務もある．すなわち，組織でのコラボレーション基盤を適切な単位に分割し，部分ごとに管理方針を変えることにより，コラボレーション活性化と企業リスク低減といった，一見トレードオフの関係にある課題の両立が可能という仮説を立てた.

4. コラボレーション基盤の分割統治

前節で述べた分割統治のアイデアに基づき，コラボレーション基盤を活用するための二つの分割を提案する．第一の分割は，コラボレーション基盤を使った情報システムを構造的に分割し，その分割された要素ごとに管理の枠組みを変える．第二の分割は，コラボレーション基盤をビジネスプロセスの道具と考え，事業の段階ごとにコラボレーション基盤の管理を分離する．この二つの分割について 4.1，4.2 にそれぞれ説明し，これらの分割を組み合わせたコラボレーション基盤の管理モデルを 4.3 に示す.

4.1 コラボレーション基盤の層ごとの分割

現在，多くの企業では，従来からハードウェアやソフトウェアを運用していた情報システム管理者が，コラボレーション基盤も管理している．コラボレーション基盤では，コンテンツを単にデータ容量，バイト数で管理するのではなく，その意味の吟味が必要であり，ハードウェアやソフトウェアの管理と本質的に異なる．ハードウェアやソフトウェアパッケージの管理負荷はその利用数やコンテンツ量に比例して増えることはなく，ハードウェアやソフトウェアの仕様範囲内であればコラボレーション基盤の活用度と無関係の場合が多い．一方，コンテンツの管理は，コンテンツの量に比例して管理の負荷が増加する．従来の情報システム管理の人材，枠組みで，コラボレーション基盤を管理しようとすると，管理スキル，管理負荷の両面で問題が発生するのである.

この問題を解決するために，米国の法学者ベンクラー [87]やレッシグ [88]が提唱している通信システムの「層（Layer)」という概念が流用できる．ベンクラー [87]によると，通信システムは次の三層に分類できる.

・物理層（Physical Layer)：
　コンピュータ，電話，通信回線，無線装置等の，物理的に情報を転送するハードウェアの層.

・論理層（Logical Layer)：
　規格，通信プロトコル，ソフトウェア等の，人間に意味のあるものをコンピュー

タに分かるように変換する層[9].

・コンテンツ層（Contents Layer）:

エンドユーザにとって意味のある情報，著作物.

表 5- 1 で示したコラボレーション基盤のリスクをこれらの層で再分類し，層ごとのリスク対策を図 5- 1 にまとめた．ここでのリスク例はすべてのリスクを網羅したものではないが，重大な被害を引き起こす危険性のある他のリスクも含め，各層で必要なリスク対策を示した．これらのリスク対策を現実的な負荷で的確に実施するための各層の管理方針を次に述べる.

（1）物理層

物理層でのリスクは，コラボレーション基盤に固有なリスクではない．したがって，この層では格納，媒介されるコンテンツの種類や，サービスの種類とは独立に管理が行えるようにする．物理層のスケーラビリティは，間接的にコンテンツ層，論理層と関係を持つが，コラボレーション基盤の物理層としての要件が決まれば物理層の管理者は独立して管理できるようにする．特定のサービス，特定のコンテンツと独立した物理層の管理方針を採用し，その自動化を進めることにより，物理層の管理負荷を削減することができる.

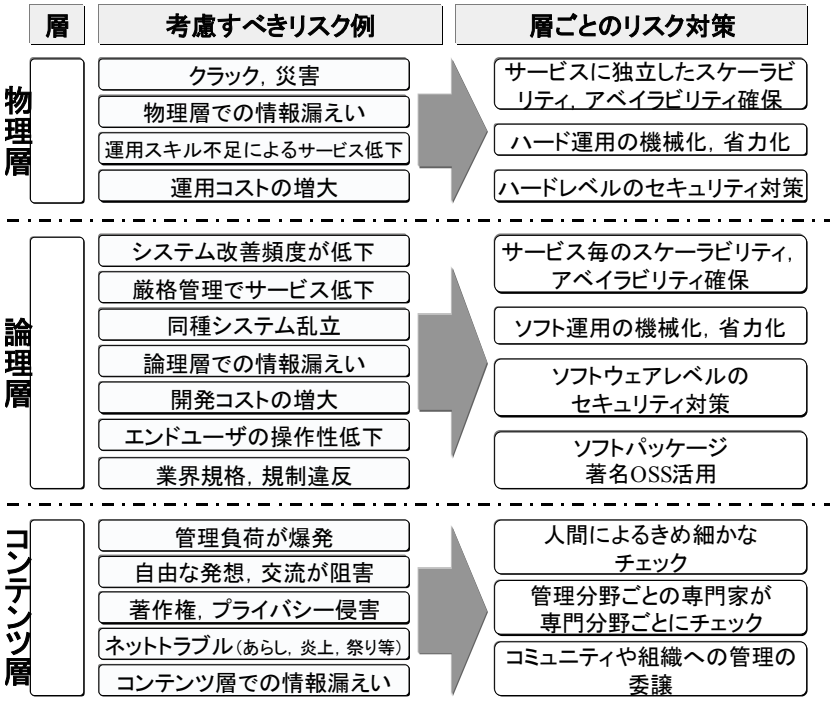

図 5- 1 コラボレーション基盤の層による分割

Figure 5-1 Collaboration System Layer Segmentation

[9] レッシグ [88]は，論理層を「コード層」と呼んでいる.

（２）論理層

論理層では，物理層の上に乗る情報システムの管理，すなわちシステム設計，ソフトウェア導入，保守等を行う．論理層によってコラボレーション基盤が提供する機能やセキュリティの強度が決まるため，この層で所定のサービスレベル，セキュリティレベルを保証するための管理が必要となる．この層の管理負荷は，サービスの種類，必要とするセキュリティやデータ保全のレベルによって管理方針が異なる．しかし，この層でも，コンテンツ層の管理と明確に分離し，コンテンツの意味に踏み込んだ管理はしない．さらに，システム管理ソフトウェア等を活用することにより管理負荷を削減できるようにする．

（３）コンテンツ層

コンテンツ層でのリスク対策は，他の層とは異なり，システム本来の目的であるコラボレーションを阻害せず，自由な発想，交流を促進する部分が加わる．このため，物理層，論理層の管理は十分にできていることを前提に，コンテンツ層ではコンテンツの管理に注力する．コンテンツ層の管理は，自由な交流，自由な発想が阻害されないように人間によるきめ細かい管理が必要であり，コンテンツのデータ量に比例した負荷を必要とする．しかし，この場合でも，コンテンツの適用範囲や特性による管理負荷の分割は可能である．例えば，あるテーマを扱うコミュニティ単位に管理を分割することによって，管理負荷を分散する．さらに，セキュリティ，特許，個人情報等のコンテンツの特性単位にコンテンツの管理者を割当て，コンテンツ層の管理をさらに分割することができる．この場合，コンテンツ層の管理者は必ずしも論理層や物理層の管理者である必要は無い．

コラボレーション基盤全体としては矛盾するようなリスク対策が，図 5- 1 のように層別したことにより一貫したリスク対策を講じることができる．

4.2　事業の段階ごとにコラボレーション基盤を分割

ベンクラーの層によって，コラボレーション基盤を分割して管理した場合でも，コンテンツ層の管理にかかる労力の総和はコンテンツ量に比例することは変わりなく，膨大になりうる．この問題を解決するため，筆者は，企業のビジネスプロセスのうち，管理負荷の高いコラボレーションが必要不可欠な段階とそうではない段階に分割することによって管理負荷が低減できると考えた．すなわち，事業の段階によっては，不特定多数の執筆者が不特定多数の人に情報発信するようなタイプのコラボレーションを必要とするため，管理負荷は必然的に高い．しかし，特定のメンバ間での情報交換に閉じることができるような事業の段階の場合は，コンテンツ管理をコミュニティやプロジェクトの単位に分割し委譲することが可能である．このように，事業の段階毎に管理方針を変えることによってコンテンツ層の管理負荷は大幅に軽減できると考えた．

企業における事業の実行段階を大きなレベルで「アイデア，マーケティング段階」，「初期適用，評価段階」，「計画，実行段階」の三段階に分割 [34]し，その段階ごとに図 5- 1 で示したコンテンツ管理のリスクを整理し，その対策を図 5- 2 のようにまとめた．段階ごとのリスク例，リスク対策は，情報セキュリティマネジメントシステム規格 [85]に書かれたセキュリティ脅威及び，それの対策例を参考にした．

本章では，製造業の製品事業におけるビジネスプロセスの例を取り上げるが，他の業種であっても容易に適用できるように一般化して説明する．

図 5- 2 事業の段階によるコンテンツ層の分割

Figure 5-2 Segmentation of Business Process Content into Layers

（1）アイデア，マーケティング段階

事業の初期段階，具体的には市場理解からアイデア，事業コンセプトをまとめるまでの段階では，自由な発想やオープンな交流が重要視され，これを実現するために不特定多数の執筆者が不特定多数の人に情報発信するようなタイプのコラボレーションが必要である [34]．このようなタイプのコラボレーションを支援する基盤をオープン型コラボレーション基盤と定義する．不特定多数の執筆者が不特定多数の人に情報発信するようなコンテンツの管理はコンプライアンス関連の潜在的な脅威も大きい．オープン型コラボレーション基盤のコンテンツ層は，機械的に管理するのではなく，コミュニティの管理者及びシステム全体の管理者がきめ細かい管理を行い，コンプライアンスに対応しつつも自由な発想，オープンな交流を実現する．

（2）初期適用，評価段階

アイデア，マーケティング段階に続いて，プロトタイピング，初期適用，評価等の，事業としてのフィジビリティスタディの段階に入る．この時点では，情報の発信者，情報の

受信者とも特定されていくが，まだ完全に固定はされていない．また，組織との対比では部課等の固定的な組織を越えたコミュニティの形成は必要であるが，コミュニティの内部ではセキュアに事業を検討するタイプのコラボレーションが必要である．このようなコラボレーションを支援する基盤をフレキシブル型のコラボレーション基盤と定義する．フレキシブル型のコラボレーション基盤では，オープン型の基盤に比べてデータの保全に対する要求は高くなる．また，コミュニティのセキュリティが重要になり，コミュニティの管理者がコンテンツ層を管理する仕掛けをコラボレーション基盤が備え，コミュニティの管理者がコンテンツ層の管理を行う．

（3）計画，実行段階

さらに事業の実行が決まった場合，製品事業でいうと，製品計画，製品開発，製造，保守等の定型的な業務を実行する段階が，固定的な組織又はプロジェクトで実施される．それらの組織，プロジェクトで必要なコラボレーションは，セキュアであることはもちろん，製品，開発/製造工程，開発組織に求められる各種業界規格や規制に対応したものである必要がある．このようなコラボレーションを支援する基盤をフォーマル型のコラボレーション基盤と定義する．フォーマル型のコラボレーション基盤においても，プロジェクトの管理者がコンテンツを管理する仕掛けを備え，プロジェクト管理者がコンテンツ層の管理を行う．

事業の各段階で必要なコラボレーション基盤の使い分けを図 5- 3 に示す．図 5- 3 では．縦軸にメンバが特定か否か，横軸に業務に依存しているか否かを表した．事業の初期段階では不特定メンバ業務非依存のオープン型を使用し，次に特定メンバでセキュアに議論できるフレキシブル型へ移行し，さらに，業務に密着したフォーマル型のコラボレーション基盤に移行していく．

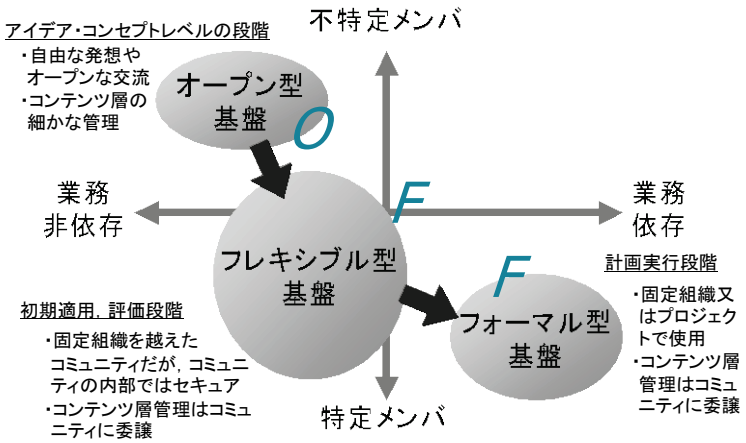

図 5- 3 コラボレーション基盤の使い分け

Figure 5-3 Collaboration Infrastructure Segmentation Map

4.3　ＯＦＦコラボレーションモデル

4.1 で述べた層による分割と，4.2 で述べた事業の段階による分割は独立に実施可能である．したがって，これらを組み合わせることにより，より有効な施策となる．これらの分割を組み合わせたコラボレーション基盤の管理モデル，ＯＦＦ（Open, Flexible, and Formal）コラボレーションモデルを表 5- 2 に示す.

このモデルでは，4.2 で述べた事業の段階でまず分割する．すなわち，各段階での要件から，コラボレーション基盤はオープン型（Open）から，フレキシブル型（Flexible）へ，さらに，フォーマル型（Formal）へと変更する．さらに，これらのコラボレーション基盤に対応して，4.1 で述べたベンクラーの各層ごとの管理方針も変えていく．このように，事業の段階での分割と層での分割を組み合わせて適切に管理していくことにより，アイデア・マーケティング段階におけるオープンなコラボレーション，初期適用，評価段階でのフレキシブルなコラボレーション，計画実行段階でのフォーマルなコラボレーションが最小限の管理負荷で，セキュアに実現できる.

表 5- 2 ビジネスプロセスと層の分割によるコラボレーション基盤の管理
（OFF コラボレーションモデル）

Table 2. Collaboration Infrastructure Administration Corresponding to Business Process and Layers（OFF Collaboration Model）

製品事業の段階 [79] [34]		各コラボレーション基盤の要件	層ごとの管理方針		
			コンテンツ層	論理層	物理層
マーケティング／アイデア	事業の要件と技術の統合	オープン型基盤（**Open**）： ✧ 利便性，開放性 ✧ 不特定な利用者が情報発信，情報参照可能 ✧ インターネットでも活用できるツール ✧ 組織全体の活用	コミュニティ管理者及びシステム全体管理者で管理．管理分野の専門家の全体横断的チェック	利便性，開放性を主眼としたソフトウェア	コンテンツ層，論理層から独立にスケーラビリティ，アベイラビリティの保証
	広くアイデアを募る				
	コラボレーションによるアイデア洗練				
初期適用評価	プロトタイピング	フレキシブル型基盤（**Flexible**）： ✧ フレキシブルなメンバによる利便性，セキュリティの両立 ✧ 業務に無関係なイントラネット用のツール ✧ 組織全体で活用	コミュニティ管理者による分散管理	開放性とセキュリティを両立させるソフトウェア	
	初期適用				
	評価				
	結果（製品開発につなげるか止めるか）				
計画実行	製品計画	フォーマル型基盤（**Formal**）： ✧ ISO-9001 や業界規格に沿ったツール．事業毎 ✧ 構成管理，トレーサビリティ ✧ 固定的なメンバ	プロジェクト管理者による分散管理	コンプライアンスを重視した事業に特化したソフトウェア	
	製品開発				
	量産				
	ライフサイクル				

5.　適用事例及び評価

本節では，４節の提案モデルを筆者の勤務する大規模ソフトウェア開発組織に適用した事例とその評価について述べる.

5.1　コラボレーション基盤の構成，使用法

適用した日立ソフトウェア事業部では，約 4,000 人の利用者がおり，2002 年にフォーマル型コラボレーション基盤，2004 年にフレキシブル型のコラボレーション基盤，2006 年にオープン型のコラボレーション基盤を運用開始した．各コラボレーション基盤の適用例を管理面での使い分けを中心に表 5- 3 にまとめた．

表 5- 3 コラボレーション基盤の適用例

Table3. Example of Collaboration Infrastructure

コラボレーション基盤（型及び使用ツール）	公開範囲	運用開始	コミュニティ作成	コンテンツ管理	認証基盤	コミュニティ外から参照	
						コミュニティ名，概要	コミュニティ内容
オープン型：OpenPNE ＋カスタマイズ	全社	2006/12～	誰でも	システム管理者+コミュニティ管理者	共通(LDAP)	可	可／不可
フレキシブル型：Groupmax Collaboration	全社	2004/06～	主任以上	コミュニティ管理者		可／不可	不可
フォーマル型：プロナビ	事業部内	2002/07～	システム管理者	プロジェクト管理者		不可	不可

以下，ＯＦＦコラボレーションモデルに従ったコラボレーション基盤をどのように使用して事業を推進しているかを説明する．

事業に関するインフォーマルなコラボレーションは，オープン型のコラボレーション基盤の OpenPNE [89]ベースの SNS を活用している．この基盤の画面例を図 5- 4 に示す．

図 5- 4 オープン型のコラボレーション基盤例

この基盤は，事業部内だけでなく日立全社にオープンな社内 SNS であり，例えば，日立全社に散らばる不特定多数の識者に対して自分の事業に関係する技術トピックや，すでに市場に出ている製品の評価等を公開し，オープンかつインフォーマルに議論することができる． SNS のコミュニティ内のコンテンツは，そのコミュニティの管理者のみならず，全社の SNS の管理者も全件チェックしている．

インフォーマルな議論から，製品開発の方向性，事業の実現性などを特定のメンバで議論する場合, コラボレーションの場は, フレキシブル型のコラボレーション基盤, Groupmax Collaboration [90]に移行する．この基盤の画面例を図 5- 5 に示す．Groupmax Collaboration でもオープン型の社内 SNS と同様，組織の壁を越えたメンバ間でコラボレーションが可能である．しかし，この場では，コンテンツの参照範囲はコミュニティに特定される．また，コミュニティの管理者が，コミュニティやコンテンツのアクセスポリシーをフレキシブルに制御でき，議論の内容によってオープンにもセキュアにもできる．このコミュニティでのコンテンツ層の管理は，システム全体ではなく，コミュニティの管理者に委譲している．

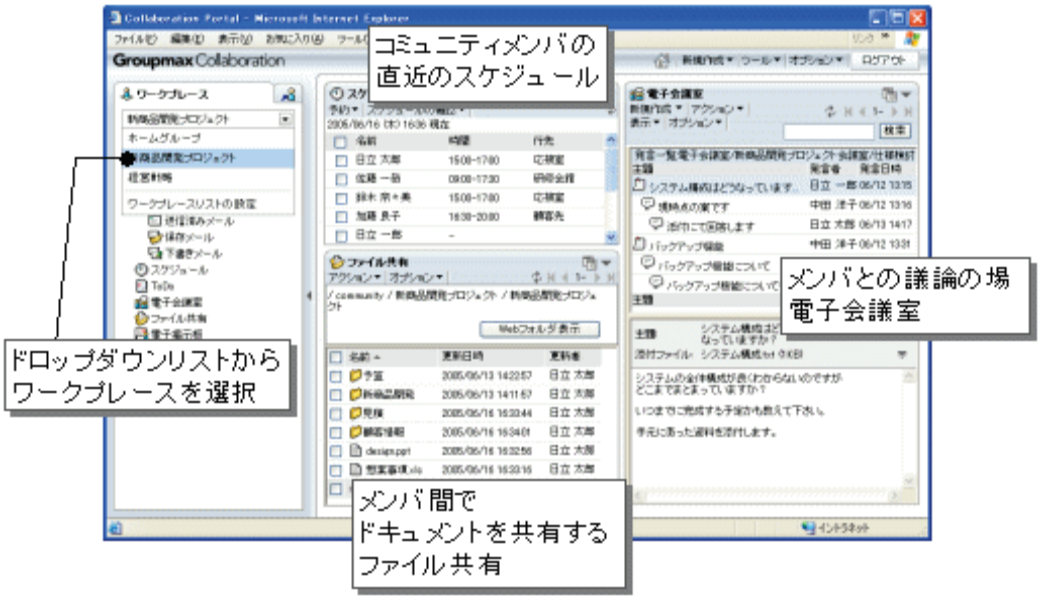

図 5- 5 フレキシブル型のコラボレーション基盤例

事業の実行が承認されたタイミングで，コラボレーション基盤も，社内の規格や，内部統制などの各種規則，規制にフォーマルに対応する必要がある．開発，製造工程やそれに対応する規格は，事業によって異なる．筆者の勤務する事業部も，この部分は，日立全社ではなく，事業部固有のビジネスプロセスに対応した，プロナビと名づけられたフォーマルなコラボレーション基盤 [91]を活用している．この基盤の画面例を図 5- 6 に示す．この

基盤は，ISO-9001:2000 [92]及び TickIT [93]に準拠した事業部の開発規格に対応したフォーマル型のコラボレーション基盤であり，製品計画からソフトウェア開発，出荷，保守までの一連の工程をサポートしている．この基盤は，事業部規格に対応した厳格なアクセスポリシーに固定されており，コンテンツ層の管理はプロジェクト管理者のタスクとしている．

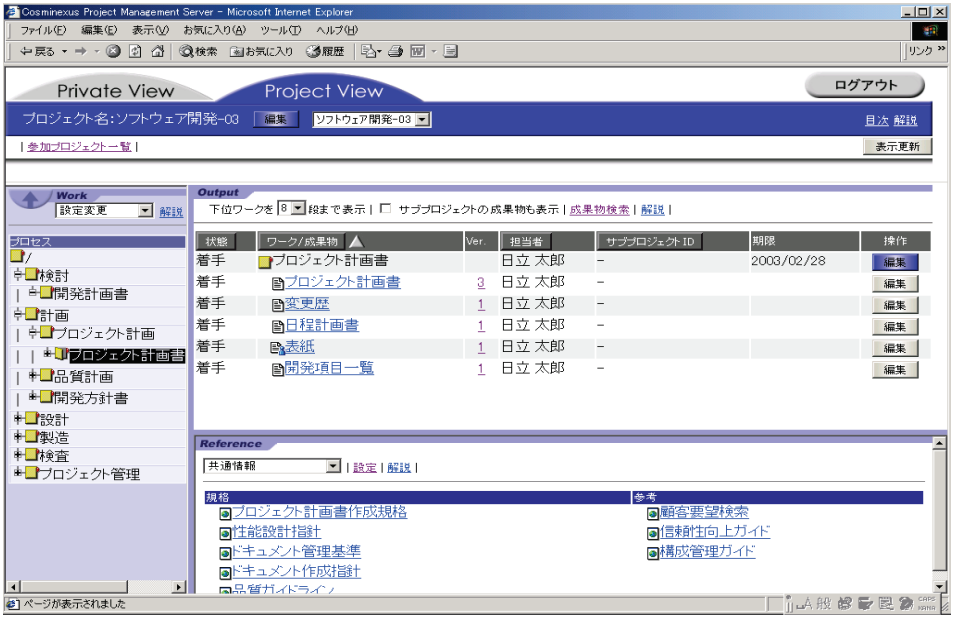

図 5- 6 フォーマル型のコラボレーション基盤の画面例

このように，コラボレーション基盤を事業の段階ごとに使い分け，一部分に管理負荷の高いオープン型のコラボレーション基盤を配置することにより，コラボレーション基盤の利便性とセキュリティを現実的な管理負荷で実現している．

5.2 提案モデルの適用結果

本項では，提案モデルの適用結果を示す．対象は，ＯＦＦモデルによるコラボレーション基盤が本格適用になった 2007 年 1 月から 12 月までの 1 年間の，筆者の勤務する事業部のコラボレーション基盤の活用及び運用データである．表 5- 3 で述べた適用事例では，オープン型基盤，フレキシブル型基盤は全社で管理するコラボレーション基盤で，フォーマル型基盤のみが事業部で管理するコラボレーション基盤である．本項のデータは，オープン型基盤，フレキシブル型基盤についても，評価対象のソフトウェア事業部の従業員が立ち上げたコラボレーション基盤のコミュニティ又はプロジェクト [10]のみを対象とした．全社対象のコラボレーション基盤の管理負荷は，仮に対象の事業部のコミュニティのみを運用した場合の想定管理負荷を運用者にヒアリングした結果を使用した．

[10]コミュニティのメンバの一部は他事業部の従業員の場合もある．

（1）　コンプライアンス対応の結果

ＯＦＦモデルを構成する，各コラボレーション基盤に起因する，セキュリティ問題，個人情報漏えい，著作権侵害事故など，コンプライアンス関連の事故は評価期間中一件も発生していない．オープン型基盤においては，全社 SNS 管理者のチェックによって，コンプライアンス関連の事故につながりかねない潜在的な問題を複数件摘出している．

（2）　コラボレーション活性化状況

コミュニティの平均寿命は約 1 年であるため, 1 年間で新規開設したコミュニティ数でコラボレーション活性化を計測した. ＯＦＦモデルを適用していなかった 2003 年では約 2400 コミュニティが新設された．これに対して，ＯＦＦモデルを適用した 2007 年では，約 4200 コミュニティが新設され，2003 年と比べて約 75%増加した．次に，ＯＦＦモデルに従ったコラボレーション基盤の型別活用状況を図 5- 7 に示す．フォーマル型が 56%，フレキシブル型が 42%，オープン型は 2%である．

また，適用した事業部のコラボレーション基盤使用対象者一人当たりの平均参加コミュニティ数は 10 以上である．コミュニティには，部課といった自分の所属する組織単位のコミュニティと，組織横断的なコミュニティがあるが，ほとんどの従業員は，この両種類のコミュニティに参加し，定型業務，非定型業務に渡って活用している．

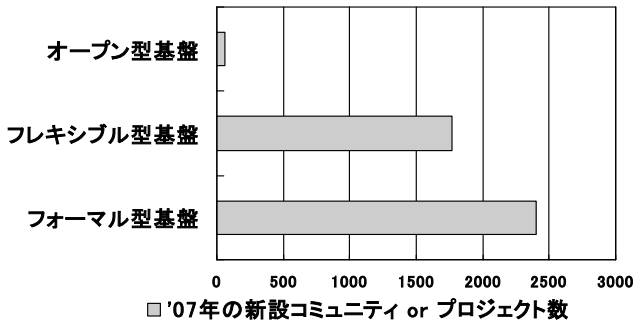

図 5- 7 コラボレーション基盤の活用状況

Figure 5-7. Status of Collaboration Infrastructure Application

（3）　コラボレーション基盤管理負荷の結果

管理コストには，運用開始段階での初期コストと定常運用時のランニングコストがある．今回の事例で使用したコラボレーションツールはすべて Web ベースのパッケージソフトウェアのため，初期コストは大きくない．したがって，ＯＦＦモデル各型のコラボレーション基盤の定常運用時における管理負荷を比較する．

ＯＦＦモデル各型のコラボレーション基盤が提供するコミュニティごとのシステム運用

者の管理負荷を図 5- 8 に示す．図 5- 8 はフォーマル型基盤のコミュニティの管理負荷を 1 としたときの相対値である． 図 5- 8 からフォーマル型のコラボレーション基盤のシステム運用者の管理負荷を 1 とすると, オープン型は 6 倍弱と高い管理負荷であることが分かる．

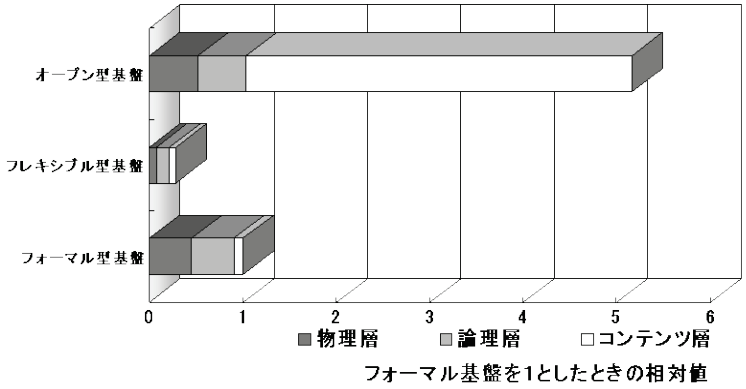

図 5- 8 コラボレーション基盤の管理負荷(運用人数／コミュニティ新設数)

Figure 5-8. Collaboration Infrastructure Administration Load
(Operation Personnel/No. of New Communities)

図 5- 8 で各型の管理で，どの層の管理負荷が高いかをみる．オープン型の負荷の 8 割はコンテンツ管理の負荷である．コンテンツ量に比例する管理負荷が高いことが確認できた．

5.3　評価と考察

適用事例及びその結果に基づき，本章で示したモデルの有効性を，コンプライアンス対応，コラボレーショ活性化，管理負荷の各面から評価，考察する．

（1）　コンプライアンス対応の評価

コンプライアンスやセキュリティに関連した問題が発生する確率の高いオープン型のコラボレーション基盤を新設したにもかかわらず，コンプライアンス関連の事故は発生しなかった．この理由として，

- フォーマル型でないコミュニティもＯＦＦモデルによりフレキシブル型とオープン型に分離
- 限られた数のオープン型のコミュニティに対して，コミュニティの管理者及び社内 SNS 管理者の二重のチェック体制で，潜在的な問題を摘出

の二点が有効に作用したと考える．

（2）　コラボレーション活性化の評価

ＯＦＦモデルを適用せずフォーマル型のコラボレーション基盤しかなかった2003年以前と比較し，ＯＦＦモデルに基づくフレキシブル型コラボレーション基盤を2004年，オープン型コラボレーション基盤を2006年にそれぞれ適用したことにより,2007年は年間で約2倍のコミュニティが新設された.

（3）　コラボレーション基盤管理負荷の評価

オープン型のコラボレーション基盤のシステム運用者のコンテンツ管理負荷は高く，仮にオープン型コラボレーション基盤の管理方法をそのままフレキシブル型の運用管理に適用，すなわち，図5- 8のオープン型のコンテンツ管理負荷をフレキシブル型の全コミュニティに当てはめた場合，システム運用者の管理負荷は現状の２０倍以上になる．したがって，フォーマルでないコミュニティをＯＦＦモデルによってフレキシブル型とオープン型に分離したことにより，抜本的にシステム運用者の管理負荷が削減できた.

以上により，コラボレーション基盤の課題に対する提案したモデルの有効性が示せたと考える.

今回の事例では，本質的に自由な発想，自由な交流が必要なオープン型のコラボレーション基盤の活用度が事業全体に占める割合は必ずしも高くないという結果が得られた．ここで，本当に現状のオープン型コラボレーションの割合が最適なものなのかどうかは必ずしも今回の結果で明らかではない．今回の事例でも，事業効果を最大にするオープン型コラボレーションの割合が前項結果の 2 倍であることはありうるし，適用する企業や，事業内容に依存して，さらに違う値が最適である可能性も高い．しかし，オープン型のコラボレーションの適用範囲を局所化することにより，管理負荷を削減でき，かつコンプライアンス関連の管理も現実的な負荷で可能になるという結果は企業や事業内容に独立に適用できると考える.

6.おわりに

本章は，コラボレーション基盤を層，また事業の段階で分割し，本質的に人間による管理が必要な部分を局所化すると共に，管理負荷を分散させ，フレキシブルなコラボレーションと，企業に求められる各種規制を両立させるＯＦＦコラボレーションモデルを提案した．本提案モデルでは，企業内の情報システムの構成要素，事業の段階ごとに必要なコラボレーション基盤の要件及び管理方針を示した.

提案したＯＦＦコラボレーションモデルを社内の事業部に適用した結果，ビジネスプロセスに適合したコラボレーションが活性化したばかりでなく，コラボレーションシステム運用の管理負荷が削減でき，かつ，情報漏えいやセキュリティ事故等のコンプライアンス系の問題を発生させなかった．これらの結果により，ＯＦＦモデルが実務に有効であることが分かった.

今後は，提案モデルを継続して評価することにより，精度の高い定量的な効果の算出，さらに効率的なコラボレーション基盤管理を行うための改善モデルなども検討する予定である．

第六章　結論

本研究は，ソフトウェア開発を行う組織を「ソフトウェア生産システム」とし，そのシステムがどのようにモデル化可能か，どのようにそのシステムの運用，すなわち，最適化，改善，統制が可能かを示した．また，このソフトウェア生産システムの運用に必要な知識の総体を「ソフトウェア生産技術」と定義し，その全体像を明確にするとともに，具体事例を通して，ソフトウェア開発を組織的かつ継続的に最適化および改善する具体的な方法を示した．本章では，全体のまとめとして，本研究で得られた成果をまとめるとともに，今後の研究に委ねる課題を示す．

1.　成果

本研究では，組織的に大規模かつ高品質なソフトウェアを開発するために必要な知識をソフトウェア生産技術と定義し，その源流を大きく経営工学とソフトウェア工学にもとめ，20 世紀初頭以降のこれまでの知識の積み重ねを概観した．この結果，従来，ソフトウェア工学の積み重ねの延長で理解されてきたソフトウェア生産技術の多くの知識が，経営工学で積み上げられた知識を前提にしていることを示した．さらに，ソフトウェア生産技術を「ソフトウェア生産技術の対象」，「ソフトウェア生産技術の業務機能」の二つに分けてモデル化し，その組織最適化および，継続的改善のモデルを示し，従来のプロジェクトレベルでの改善モデルではなく，組織レベルの改善モデルを示すことができた．

三章では，筆者の勤務している，(株)日立製作所ソフトウェア事業部の生産技術活動を例に，「ソフトウェア生産技術の対象」，「ソフトウェア生産技術の業務機能」および，組織最適化/継続的改善のモデルの実際の活動，効果を示し，その実用性を示した．

四章では，ソフトウェア生産技術によるソフトウェア開発組織全体の定量化事例および成果を示した．従来の生産性，すなわち成果量÷コストに加え，組織の俊敏さを計測するメトリクスとして多くの産業分野で広く活用されている回転率をソフトウェア開発においても導入できることを示し，このソフトウェア開発回転率メトリクスが，多様なプロセスモデルに従ったソフトウェア開発プロジェクトを多数持つようなソフトウェア開発組織でも適用可能であることを明確にした．

第五章では，ソフトウェア製品開発の組織での知識の蓄積方法および事例を述べた．大規模ソフトウェア開発組織で必要なプロジェクトをまたがった知識の共有と，多様な知的財産権に由来する情報保全への要求を両立させる情報管理モデル，ＯＦＦコラボレーションモデル (Open, Flexible, and Formal collaboration model)を提案し，適切な知識共有と分離を実現する知識共有インフラと，製品開発の各フェーズでどのように使い分けるかを示した．

2. 今後の課題

本研究では，ソフトウェア生産技術の対象と業務機能という全体の枠組みは示したが，その枠組みに従った，個々の技術については網羅できていない．今後の研究で，ソフトウェア生産技術として不特定組織で有効な技術について形式知化していく予定である．

また，本研究では，ソフトウェア開発組織に着目し，そのソフトウェア生産技術がどのようなものか，どのように活用可能かを示した．しかし，昨今は，ある組織内でソフトウェアを開発するだけでなく，ソフトウェアの一部を他組織への開発委託，複数企業の共通開発というように，一組織に閉じない開発形態も多くなってきている．このような形態は，ハードウェア製造における，サプライチェーンに対応しているが，ソフトウェアの場合，一般に発注から検収までの期間が長く，発注側の受注側への制御がハードウェア製造の場合に比べて困難という違いも多い．また，今後は，フォーマルな組織間の発注/受注という関係だけでなく，インフォーマルなソフトウェア開発コミュニティとなどとの関連も含めて考える必要もある．このような，ソフトウェア開発におけるサプライチェーンというソフトウェア固有の生産技術も今後の課題である．

本研究では組織化の最終形態としてインスタンスレベルの共通化を定義した．現在，企業で使用する計算機リソースの集中化がクラウドという名前のもとに進んでおり，ソフトウェア開発に必要な計算機リソースも今後，企業内に設置したプライベートクラウドや，インターネットを介したクラウドが提供するサービスを利用する機会が増えてくると考える．このような開発環境が一般になったときに，現在，ある企業に閉じて動いているソフトウェア生産技術は，企業の壁を越えてどのようなサービスとして提供できるのか，そのようなサービスでは，どのように，知的財産権の確保と使用者(使用企業)の利便性を両立させるのかということも今後の課題である．

謝辞

本研究の全般に関し，ご指導を賜りました，静岡大学情報学部情報科学科 酒井三四郎教授に，心から深く感謝申し上げます．酒井先生には，大学院後期課程に入学以前から研究全般についてご指導をしていただきました．

本研究の実施，並びに本論文を執筆するにあたり，適切なご助言とご指導を頂きました，静岡大学情報学部 市川照久特任教授に心より感謝致します．市川先生には，情報処理学会 IS 教育委員会を始め多くの場で，情報処理学会フェロー神沼靖子先生とともにご指導をいただくとともに，今回の研究のきっかけを作っていただきました．

本論文の執筆にあたり，適切なご助言とご指導を頂きました，静岡大学情報学部情報科学科 塩見彰睦教授，渡辺尚教授，青木徹准教授，松澤芳昭助教に心より感謝致します．プレゼンテーションでの指摘のみならず，論文の書き方レベルから研究の核心部分にいたるまで懇切丁寧にご指導をいただきました．

本研究の基となったソフトウェア生産技術，コラボレーション基盤関連の成果は，（株）日立製作所ソフトウェア事業部の生産技術部門，品質保証部門の礎を築いた，芝田寛二氏，片岡雅憲氏，飯野守夫氏，横山陽一氏，堀内純孝氏をはじめ多くの先輩方のご指導と，共に研究を行った，谷田耕救氏，大島真幸氏，大場みち子博士（現はこだて未来大学）との共同作業によるものです．多くの協力を頂き，厚く感謝いたします．

本論文第四章の Validation モデルを使った回転率に関して品質関連の情報提供していただいた，（株）日立製作所ソフトウェア事業部の品質保証部の小泉博靖氏（現本社品質保証本部），梯雅人氏，野上敬文氏に深く感謝します．また，この章の内容全般に渡り深い知見に基づいたご意見をいただいた，河野善彌先生（元（株）日立製作所戸塚工場，元埼玉大学）に感謝します．

本論文第五章のコラボレーション関連の情報提供にあたり，（株）日立製作所執行役常務の大野治博士，（株）日立製作所情報システム事業部の高橋純一氏，藤田智巳氏，（株）日立総合計画研究所の皆様の多大なるご協力を得ました．特に大野博士にはデータ提供を承諾していただいたにのみならず，ソフトウェア生産技術全般に渡るご指導や学位取得への道筋まで立てていただきました．

本研究にあたり，機会を与えて頂いた（株）日立製作所ソフトウェア事業部の小塚潔氏（現（株）日立情報システムズ），中村孝男氏（現（株）日立情報システムズ），および坂上秀昭氏（現（株）日立ソリューションズ），阿部淳氏，尾山壮一氏，渡邉友範氏に感謝するとともに，研究や論文執筆で協力頂いた生産技術部の管正志部長を始め生産技術部の同僚の皆様に感謝いたします．

文献目録

[1] IEEE Computer Society, ソフトウェアエンジニアリング基礎知識体系 ―SWEBOK.: オーム社, 2003.

[2] Project Management Institute, A Guide to the Project Management Body of Knowledge: Official Japanese Translation(プロジェクトマネジメント　知識体系ガイド　PMBOK ガイド), 4th ed.: Project Management Inst, 2009.

[3] 松本吉宏, "日本のソフトウェア工場," in ソフトウェア工学大事典.: 朝倉書店, 1998, pp. pp.1082-1094.

[4] V. R. Basili, G. Caldiera, and H. D. Rombach, "経験工場," in ソフトウェア工学大事典., 1998, pp. 182-190.

[5] W. S. Humphrey, ソフトウェアプロセス成熟度の改善.: 日科技連出版社, 1991.

[6] K. Beck and et al. (2001) Manifesto for Agile Software Development. [Online]. http://agilemanifesto.org/

[7] M. Poppendieck and T. Poppendieck, リーン開発の本質 ソフトウェア開発に活かす 7 つの原則.: 日経 BP 社, 2008.

[8] 大野耐一, トヨタ生産方式―脱規模の経営をめざして.: ダイヤモンド社, 1978.

[9] M.ビードル K.シュエイバー, アジャイルソフトウェア開発スクラム.: ピアソンエデュケーション, 2003.

[10] Hirotaka Takeuchi and Ikujiro Nonaka, "New New Product Development Game," Harvard Business Review Article 1986 Jan., 1986.

[11] K. Beck, XP エクストリーム・プログラミング入門―変化を受け入れる　第二版.: ピアソンエデュケーション, 2005.

[12] W. S. Humphley, "プロセス成熟度モデル," in ソフトウェア工学大事典., 1998, p. 1295.

[13] M. Cusumano, ソフトウエア企業の競争戦略.: ダイヤモンド社, 2004.

[14] J. E. Tomayko, "ソフトウェア工学におけるマイルストーン," in ソフトウェア工学大事典., 1998, pp. 699-708.

[15] F. P. Brooks, 人月の神話　―狼人間を撃つ銀の弾はない.: ピアソンエデュケーション, 2002.

[16] James E. Tomayko. (1988) Computers in Spaceflight, The NASA Experience. [Online]. http://www.hq.nasa.gov/office/pao/History/computers/Compspace.html

[17] 金子則彦, 旅人をつなぐ“マルスシステム”開発ストーリー.: アイテック , 2005.

[18] 日経 BP 社, "1965 年 三井銀行、国内初のオンライン・バンキング・システム (特集 伝説のプロジェクト 今に通じる不変の成功哲学)," 日経コンピュータ, no. 617, pp. 40-43, Jan. 2005.

[19] G. Salvendy 編集, IE ハンドブック.: 日本能率協会, 1986.

[20] F. W. Taylor, 科学的管理法. : 産能大学出版部, 1969.

[21] 法雲俊邑, 経営工学.: オーム社, 1989.

[22] 竹村伸一, システム技法ハンドブック.: 日本理工出版会, 1981.

[23] J. D. C. Little, "A Proof of the Queueing Formula L = λ W," Operations Research, vol. 9, pp. 383-387, 1961.

[24] A. V. Feigenbaum, Total Quality Control.: McGraw-Hill, 1961.

[25] 石川馨, 品質管理入門, 3rd ed.: 日科技連出版社, 1989.

[26] Eliyahu M. Goldratt, ザ・ゴール — 企業の究極の目的とは何か.: ダイヤモンド社, 2001.

[27] M. Poppendieck and T. Poppendieck, リーンソフトウエア開発~アジャイル開発を実践する 22 の方法.: 日経 BP 社, 2004.

[28] M. E. Fagan, "Design and code inspections to reduce errors in program development," IBM System Journal, Vol.15 No.3 1976.

[29] H.D. Mills, M. Dyer, and R.C. Linger, "Cleanroom Software Engineering," IEEE Software 4 (5) 1987.

[30] 佐藤武久, 金藤栄孝, and 大槻繁, ソフトウェアクリーンルーム手法—高品質ソフトウェア開発パラダイム.: 日科技連出版社, 1997.

[31] A. J. Albrecht, Measuring Application Development Productivity.: IBM, 1979.

[32] C. Jones, ソフトウェア開発の定量化手法.: 共立出版, 1988.

[33] 除村健俊, "IBM の製品開発体系 IPD におけるプロジェクトの考え方(我が社の PM 事例)," プロジェクトマネジメント学会誌 8(1) 2006.

[34] 日本 IBM IPD 研究チーム 広瀬 貞夫 (監修), IPD 革命—製品開発の変革ソリューション 売れる・儲かる・満足を与える製品開発の仕組み.: 工業調査会, 2003.

[35] M. Cusumano, "ソフトウェア工場," in ソフトウェア工学大事典 (Marciniak, John J.編）.: 朝倉書店, 1998, pp. pp.748-761.

[36] C. Fernstrom and R. Rockwell, "EUREKA ソフトウェア工場," in ソフトウェア工学大事典.: 朝倉書店, 1998, pp. pp.1479-1481.

[37] J.Greenfield. (2005, May) マイクロソフト社. [Online]. http://msdn.microsoft.com/ja-jp/library/aa480032.aspx

[38] G. Karner, "Use Case Points - Resource Estimation for Objectory Projects," Objective Systems 1993.

[39] 藤井拓，細川亮二，木村めぐみ, "機能規模測定手法 COSMIC 法を用いた画面生産性の分析事例," ソフトウェアエンジニアリングシンポジウム 2009 2009.

[40] B. W. Boehm, Software Engineering Economics. Englewood Cliffs, NJ.: Prentice Hall, 1981.

[41] 誉田直美, ソフトウェア品質会計—NEC の高品質ソフトウェア開発を支える品質保証技術.: 日科技連出版社, 2010.

[42] 全社 SWQC 活動調整委員会 , ソフトウェアの総合的品質管理 NEC の SWQC 活動.: 日科技連出版社, 1990.

[43] 橋本弥一郎, 平島俊一, 古賀恵子, 横山陽一, and 森文彦, "ソフトウェア品質評価システム“ＳＱＥ”," 日立評論, 巻: ６８号 1986.

[44] M. Ikoma, K. Koga, Y. Hashimoto, and F. Mori, "Software Quality Estimation System 'SQE'," Strasbourg France, Proceedings of 6th International Conference on Reliability and Maintainability 1988.

[45] 芦田典子, 居駒幹夫, and 古賀恵子, "設計段階での大規模ソフトウェア品質管理方式," in 第 10 回ソフトウェア生産における品質管理シンポジウム., 1990.

[46] E. Yourdon, ICSE peopleware panel session., May 2007. [Online]. http://www.yourdonreport.com/index.php/2007/05/29/icse-peopleware-panel-session/

[47] C. Jones, ソフトウェア品質のガイドライン.: 構造計画研究所, 1999.

[48] 独立行政法人 情報処理推進機構, ソフトウェア開発データ白書 2010-2011.: 独立行政法人 情報処理推進機構, 2010.

[49] 野中郁次郎 and 竹内弘高, 知識創造企業.: 東洋経済新報社, 1996.

[50] R. McFeeley, IDEAL: A Users Guide for Software Process Improvement. Pittsburgh, Pennsylvania, USA: Software Engineering Institute, Carnegie Mellon University, 1996.

[51] SEI. (2010, Sep.) Process Maturity Profile. [Online]. http://www.sei.cmu.edu/cmmi/casestudies/profiles/pdfs/upload/2010SepCMMI.pdf

[52] 居駒幹夫. (2003) 製品ビジネスにも CMMI は必要。では十分か。, SEPG Japan 2003. [Online]. http://www.jaspic.org/event/2003/SepgJapan/1B2.pdf

[53] V. Basili, Software Modeling and Measurement: The Goal/Question/Metric

Paradigm.: University of Maryland, CS-TR-2956, UMIACS-TR-92-96, 1992.

[54] K. Pohl, G. Bockle, and F. Linden, ソフトウェアプロダクトラインエンジニアリング—ソフトウェア製品系列開発の基礎と概念から技法まで.: エスアイビーアクセス , 2009.

[55] E. Raymond, 伽藍とバザール.: 光芒社, 1999.

[56] P. M. Watt-Morse, "データ権," in ソフトウェア工学大事典., 1998, pp. 928-932.

[57] P. Senge, フィールドブック 学習する組織「5 つの能力」 企業変革をチームで進める最強ツール.: 日本経済新聞社, 2003.

[58] B. W. Boehm, Software Cost Estimation with COCOMO II.: Prentice Hall, 2000.

[59] M. Cusumano, 日本のソフトウェア戦略—アメリカ式経営への挑戦.: 三田出版会, 1993.

[60] ISO, ISO/IEC 9126-1:2001 Software engineering — Product quality — Part 1: Quality model., 2001.

[61] Philippe Kruchten, ラショナル統一プロセス入門, 2nd ed.: ピアソンエデュケーション, 2001.

[62] B. W. Boehm, "A spiral model of software development and enhancement," Computer, vol. 21, no. 5, pp. 61 - 72 , May 1988.

[63] Eliyahu M. Goldratt, クリティカルチェーン—なぜ、プロジェクトは予定どおりに進まないのか?.: ダイヤモンド社, 2003.

[64] H. D. Rombach, V. R. Basili, and R. W. Selby, Experimental Software Engineering Issues: Critical Assessment and Future Directions.: Springer-Verlag, 1993.

[65] 芝田寛二, "ソフトウェア製品生産管理：ソフトウェア製品の生産計画と工程管理," 情報処理 21(10), 1980.

[66] M. Ikoma, "Reliability Data Collection and Its Application in Hitachi," in Proc. of 3rd Int. Conf. on Software Quality. Lake Tahoe, Nevada, USA, 1993.

[67] 芝田寛二, 横山陽一, "総合ソフトウェア生産管理システム"CAPS"," 日立評論 12 月号, 1980.

[68] 片岡雅憲, ソフトウェアの品質管理 (日科技連ソフトウェア品質管理シリーズ)., 1986, ch. 3, p. 93.

[69] 花田収悦, "ソフトウェア生産性の評価と管理," 1980.

[70] D. J. Anderson, "アジャイルソフトウェアマネジメント," 2006.

[71] 斎藤静樹編著, "財務会計—財務諸表分析の基礎," 2000.

[72] 前田久喜, "在庫圧縮の進め方ー物の流れを切り口としたロジスティクス革新," 1998.

[73] 菅野文友, ソフトウェア・エンジニアリング.: 日科技連出版社, 2000.

[74] J. Denne and M. Cleland-Huang, "The Incremental Funding Method, A Data Driven Approach to Software Development," vol. 21, no. 3, 2004.

[75] IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology., 1990.

[76] B. W. Boehm, "Verifying and Validating Software Requirements and Design Specifications," vol. 1, no. 1, 1984.

[77] C. Larman and V. R. Basili, "Iterative and Incremental Development A Brief History," IEEE Computer, vol. 36, no. 6, pp. 47-56, 2003.

[78] 保田勝通, ソフトウェア品質保証の考え方と実際　ーオープン化時代に向けての体系的アプローチ.: 日科技連出版社, 1995.

[79] D. L. Newbold and M. C. Azua, "A model for CIO-led innovation," IBM Systems Journal, vol. 46, no. 4, pp. 639-650, 2007.

[80] インプレス R&D, インターネット白書 2007., 2007.

[81] 諸橋政幸, 永井康彦, 荒井正人, 手塚悟, "ISO15408/ISO27001 統合型システムセキュリティ設計技法の提案," 情報処理学会論文誌, vol. 48, no. 11, pp. 3520-3531, 2007.

[82] 山井成良, 岡山聖彦, 繁田展史, 宮下卓也, "大規模組織における POP before SMTP に基づく管理の容易な電子メールシステム運用方法," 情報処理学会論文誌, vol. 46, no. 4, pp. 1041-1050, 2005.

[83] 堂前清隆, "インターネット生活向上委員会：Wiki で情報共有," 情報処理学会会誌, vol. 45, no. 5, 2004.

[84] 高橋秀和. (2007) ITPRO. [Online]. http://itpro.nikkeibp.co.jp/article/OPINION/20070116/258785/

[85] ISO, ISO IEC 13335-1（JIS Q 13335-1 2006） Information technology -- Security techniques -- Management of information and communications technology security., 2004.

[86] 日経 BP 社, "特集 1　エンタープライズ 2.0 Web が開く新基幹システム," 日経コンピュータ, Apr. 2006.

[87] Y. Benkler. (2006) The Wealth of Networks. [Online]. http://www.benkler.org/wealth_of_networks/index.php?title=Download_PDFs_of

_the_book

[88] L. Lessig, コモンズ.: 翔泳社, 2002.

[89] (2008) OpenPNE 公式 SNS. [Online]. http://www.openpne.jp/

[90] (株)日立製作所. (2008) Groupmax Version 7. [Online]. http://www.hitachi.co.jp/Prod/comp/soft1/groupmax/info/index.html

[91] 中津望，石崎裕，中村満明，石川貞裕，建部清美, "プロジェクト支援ツール”プロナビ”を用いた進捗管理の強化施策," プロジェクトマネジメント学会誌, vol. 8, no. 2, pp. 37-42, Apr. 2006.

[92] ISO, ISO 9001:2000　Quality management systems.: ISO, 2000.

[93] The British Standards Institution. (1995) TickIT. [Online]. http://www.tickit.org/

[94] 東基衛, "ソフトウェア工学の課題と経営工学的アプローチ(ソフトウェアの生産をめぐって)," 日本経営工学会誌 38(6B) 1988.

[95] 貝原俊也 小林英三, "製造業復活の理論—制約理論（TOC）," システム制御情報学会誌 2002 年 10 月号 2002.

[96] 原田晃 et al., "ファンクションポイント法を応用した早期見積技法の提案とそのシステム化," 電子情報通信学会論文誌 D, Vol.J89-D, No.4 2006.

[97] 遠山亮子・野中郁次郎, "「よい場」と革新的リーダーシップ：組織的知識創造についての試論," 一橋ビジネスレビュー 2000 Sum.-Aut., 2000.

[98] Matsumoto 編, Japanese Perspective on Software Engineering.: Addition Wesley, 1989.

[99] R.E. Fairley, Software Engineering Concepts.: McGraw Hill, 1985.

[100] Robert B. Grady and Deborah L. Caswell, Software Metrics:Establishing a company-wide program. Englewood Cliffs, NJ: Prentice Hall, 1988.

[101] 小笠原秀人、中野一男、田中史朗, "ソフトウェア開発プロセスの評価と改善," 東芝レビュー VOL56,No11, 2001.

[102] 平山雅之、小島昌一、野口国雄, "ソフトウェア・エンジニアリングの動向と当社の取組み," 東芝レビュー。VOL56,No11, 2001.

[103] 有田裕一，中山典保，粟田豊, "開発プロセスの可視化とプロジェクト マネジメント," FUJITSU 2005-11 月号 (VOL.56, NO.6), 2005.

[104] 古垣 幸一，高木 徹，坂田 晶紀，岡山 大輔, "トヨタ生産方式の導入によるソフトウェア開発プロセスの革新," FUJITSU2005-11 月号 (VOL.56, NO.6), 2005.

[105] 古山恒夫，菊地奈穂美，安田守，鶴保征城, "ソフトウェア開発プロジェクトの遂行に影響を与える要因の分析," vol. 48, no. 8, pp. 2608-2619, 2007.

[106] 齊藤涼子，沖山智，平井宣, "開発プロセスの標準化と Web アプリケーション 開発への対応," FUJITSU 2006-1 月号 (VOL.57, NO.1), 2006.

[107] 野中誠, "組込みソフトウェア特性に基づくプロジェクト構築(組込みソフト産業の実態と開発の課題) ," 情報処理　Vol.46, No.6, pp. pp.684-690, 2005.

[108] 渡辺純, 丸山富子, "プロセス重視スタイルによるソフトウェア 開発の工業化への取組み," FUJITSU 2009-11 月号 (VOL.60, NO.6), 2009.

[109] 津田道夫, "基幹情報システム開発のための生産技術及び見積技術に関する研究," 2008.

[110] B. W. Boehm and R. Turner, Balancing Agility and Discipline: A Guide for the Perplexed.: Addison-Wesley Professional, 2003.

[111] 居駒幹夫 and 谷田耕救, "コラボレーション活性化と企業活動の適正化を両立させる企業情報システムモデル," 情報処理学会論文誌, vol. 50, no. 2, pp. 651-658, 2009.

[112] 居駒幹夫, 大島真幸, 谷田耕救, 大場みち子, and 酒井三四郎, "ソフトウェア開発プロジェクト、開発組織のアジリティ計測方法の提案," 研究報告 - ソフトウェア工学（SE） , vol. 2008, no. 93, pp. 17-24, Sep. 2008.

[113] M. Ikoma, M. Ooshima, T. Tanida, M. Oba, and S. Sakai, "Using a Validation Model to Measure the Agility of Software Development in a Large Software Development Organization," in ICSE Companion Volume.: IEEE, 2009, pp. 91-100.

図表目次

第二章

図

図 2-1 ソフトウェア生産技術のスコープ 10
図 2-2 ソフトウェア工学の知識の分類例 14
図 2-3 GQM の構造 19
図 2-4 ソフトウェア工学での組織の知識と関連する工学の流れ 23
図 2-5 ソフトウェア開発組織におけるソフトウェア生産技術の体系 24
図 2-6 本研究で提案するソフトウェア開発組織の改善モデル 36

表

表 2-1 ソフトウェアプロセス成熟度の 5 段階とキープロセスエリア 17
表 2-2 ソフトウェア生産技術の業務機能と対象 25
表 2-3 ハードウェアとソフトウェアの生産技術の対象 25
表 2-4 成果物の分類 26
表 2-5 開発支援の分類 27
表 2-6 ソフトウェア開発プロセスモデルの分類 29
表 2-7 ソフトウェア開発組織における知識例 30
表 2-8 改善目標設定の考慮すべき項目 34
表 2-9 組織化の分類 37
表 2-10 ソフトウェア生産技術で必要な定量化例 38
表 2-11 ソフトウェア生産技術の定着化活動 40
表 2-12 知的財産権の保護のための施策一覧 41

第三章

図

図 3-1 ソフトウェア製品開発組織の典型的なプロセス構成 44
図 3-2 ソフトウェア生産システムとしてのソフトウェア開発プロセス 44
図 3-3 ソフトウェア製品の種類と出荷後フォールト数の関係 48
図 3-4 ソフトウェア開発プロジェクト規模の分類 48
図 3-5 プロジェクト規模と出荷後のフォールト数との関連 48
図 3-6 日立ソフトウェア事業部における基準値の構成と要因（芝田 [65]） 50
図 3-7 複数プラットフォームサポート時のプロジェクト構成 52
図 3-8 一般的なソフトウェア製品と開発プロジェクトの関係モデル 53
図 3-9 稼働しているソフトウェア製品の構成例 53
図 3-10 構造的な製品体系や再利用プロジェクトを考慮したモデル 54

図 3- 11 組織共通化する前の構成管理環境 57
図 3- 12 組織共通化した構成管理環境 58
図 3- 13 プロジェクトナビゲーションシステム 60

第四章

図

図 4- 1 生産性と開発期間が矛盾するプロジェクト例 65
図 4- 2 ソフトウェア開発における回転率の例 67
図 4- 3 VALIDATION モデル（ソフトウェア構成項目ごとの状態遷移図） 69
図 4- 4 ウォーターフォールモデルでのソフトウェア構成項目の状態遷移例 71
図 4- 5 反復的に VALIDATION するプロセスでのソフトウェア構成項目の状態遷移例 71
図 4- 6 V&V の適用パターン 72
図 4- 7 ソフトウェア開発回転率の推移比較 76
図 4- 8 生産性の推移比較 77
図 4- 9 プロジェクト開発期間の分布 78
図 4- 10 プロジェクト開発期間の累積分布 78
図 4- 11 グループ B のフィールドでの障害数推移 80
図 4- 12 ソフトウェア FAULT に適用した VALIDATION モデル 83
図 4- 13 S 字型成長曲線を使ったフォールト数のソフトウェア開発回転率測定 84

表

表 4- 1 生産技術，品質管理の重点項目 74
表 4- 2 計測したグループとその特徴 76
表 4- 3 プロジェクト単位でのソフトウェア開発回転率計測例 82

第五章

図

図 5- 1 コラボレーション基盤の層による分割 89
図 5- 2 事業の段階によるコンテンツ層の分割 91
図 5- 3 コラボレーション基盤の使い分け 92
図 5- 4 オープン型のコラボレーション基盤例 94
図 5- 5 フレキシブル型のコラボレーション基盤例 95
図 5- 6 フォーマル型のコラボレーション基盤の画面例 96
図 5- 7 コラボレーション基盤の活用状況 97
図 5- 8 コラボレーション基盤の管理負荷(運用人数／コミュニティ新設数) 98

表

表 5- 1 コラボレーション基盤の課題とリスク例 87

表 5- 2 ビジネスプロセスと層の分割によるコラボレーション基盤の管理 (OFF コラボレーションモデル) 93
表 5- 3 コラボレーション基盤の適用例 94

索引

3
3M, 25
4
4M, 25
A
Albrecht, A. J., 15
Anderson, D. J., 66
B
Basili, V., 4, 18, 65
Beck, K., 4, 13
Blog, 2, 85, 86
Boehm, B. W., 16, 26
Brooks, F. P., 14, 15
C
CMM, 17, 18, 31, 32
CMMI, 17, 18, 31, 32, 33, 34
COCOMO, 16, 26
COTS, 29
CPM, 13, 29
Cusumano, M., 4, 5
D
Denne, J., 65, 72
E
eXtreme Programming, 4, 13
F
Fagan, M. E., 15
Feigenbaum, A. V., 13
G
GQM, 18, 19, 32, 33, 34
Groupmax, 94, 95
H
Humphrey, W. S., 4, 5, 15, 17, 18
I
IDEAL, 18
IEEE, 69
IFM, 72
IPA, 16, 72, 73
ISO, 27, 34, 93, 96
J
Jones, C., 16
L
Linux, 20
Little, J. D. C., 13
LOC, 38, 67
M
Mills, H. D., 15
O
OFF コラボレーションモデル, 2, 7, 85, 93, 94, 99, 101
OpenPNE, 94
OR, 13, 22, 29
P
PDCA, 13, 17, 18, 32, 35, 50
PERT, 13
PMBOK, 3
Poppendieck, M., 4, 13, 66
Q
QCD, 10, 13, 28, 31, 33, 36, 37, 38, 46
R
RAID, 59
Raymond, E., 20, 21
S
Scrum, 4
SEI, 18
Senge, P., 21
SEPG, 18
SNS, 2, 85, 86, 94, 95, 97, 98
SPL, 19
Sutherland, J., 4
T
Taylor, F. W., 4, 12
TickIT, 96
TOC, 13, 29, 32, 66
TQC, 40
U
UML, 52, 53, 54
UNIX, 50
V
V&V モデル, 63, 68, 69, 70, 81
Validation, 63, 64, 68, 69, 70, 71, 72, 73, 74, 75, 78, 79, 80, 81, 82, 83, 103
Verification, 68
W
WBS, 60
Wiki, 2, 85, 86
Windows, 50, 58
Y
YAGNI, 4, 13
あ
アイデア，マーケティング段階, 91
アジャイルソフトウェア開発手法, 4, 13, 19, 22, 41, 71, 72

アジリティ, 63, 74
暗黙知, 9, 21, 22, 24, 30, 37
い
石川馨, 13, 25
インクリメンタルモデル, 29, 75
う
ウォーターフォールモデル, 29, 70, 71, 78, 81
運用コスト, 39, 87
え
営業秘密, 41
エボリューショナリーモデル, 29
エンピリカルソフトウェア工学, 31, 33
お
大野耐一, 103
オープン型コラボレーション基盤, 91, 99
か
改善, 1, 5, 6, 10, 12, 13, 14, 17, 18, 19, 21, 24, 31, 32, 33, 34, 35, 36, 38, 40, 41, 43, 45, 46, 50, 55, 63, 64, 65, 68, 77, 79, 81, 85, 87, 100, 101
回転率, 1, 7, 63, 64, 66, 67, 68, 70, 71, 72, 73, 74, 75, 77, 79, 80, 81, 82, 83, 84, 101, 103
開発規格, 28, 57, 96
開発期間, 38, 63, 64, 65, 66, 69, 73, 74, 75, 77, 78, 79, 80, 81
開発効率, 50, 54, 55, 56
開発支援, 1, 5, 6, 25, 27, 28, 35, 41
　ツール, 3, 25, 27, 28, 35, 46
開発ステップ, 55, 56
学習する組織, 21
拡販プロセス, 44
片岡雅憲, 54, 103
伽藍とバザール, 20
管理コスト, 37, 97
管理モデル, 2, 7, 85, 88
き
基準値, 38, 47, 49, 50
季節変動, 81
機能設計, 28
共通化, 28, 31, 37, 39, 46, 56, 57, 58, 60, 102
共同化, 31, 37, 39
規律・統制, 1, 6, 24, 40, 41, 43, 45, 46, 59
く
クラウド, 27, 102
け
経営工学, 1, 3, 4, 5, 6, 9, 10, 11, 12, 13, 17, 20, 22, 24, 25, 41, 42, 81, 101
計画，実行段階, 91, 92
経験工場, 4, 18
形式知, 6, 9, 21, 24, 30, 37, 41, 59, 102
計測間隔, 74, 75
結果指標, 38, 54
こ
構成管理, 19, 27, 36, 46, 56, 57, 58, 59, 79, 93
工程, 3, 4, 12, 16, 26, 27, 28, 29, 38, 47, 59, 60, 69, 74, 92, 95
コーディング, 26, 28, 30, 75
顧客回転率, 66, 73
コミュニティ, 20, 29, 88, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 102
コラボレーション, 2, 21, 27, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 103
　基盤, 21, 27, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 103
　ツール, 86, 88, 97
コンカレント QA, 75
コンテンツ, 59, 85, 86, 88, 89, 90, 91, 92, 94, 95, 98, 99
コンテンツ層, 89, 90, 91, 92, 93, 95, 96
コンプライアンスリスク, 86
さ
サイクルタイム, 63
再利用, 9, 18, 19, 21, 27, 33, 36, 45, 46, 50, 51, 52, 53, 54, 55, 56
作業効率, 51
作業対象ステップ, 51, 55
サプライチェーン, 4, 13, 102
し
仕掛かり, 66, 67, 68, 70, 71, 72, 73, 74, 75, 81, 82, 83, 84
事業の段階, 2, 85, 88, 90, 91, 93, 96, 99
システム工学, 3, 13, 22, 24, 81
芝田寛二, 46, 49, 50, 103
社会的責任, 21, 22, 41
出荷プロセス, 44, 56
俊敏さ, 2, 7, 63, 64, 101
詳細設計, 28
状態遷移, 69, 70, 71
情報管理モデル, 2, 7, 101
情報システム, 43, 85, 86, 88, 90, 99, 103

情報セキュリティ, 87, 91
情報漏えい, 86, 87, 97, 99
初期適用，評価段階, 91, 93
す
推進指標, 38, 54
スクリーニング, 35, 56
ステークホルダ, 27, 79
せ
成果物, 1, 6, 13, 18, 25, 26, 27, 28, 30, 32, 33, 35, 36, 38, 39, 41, 44, 45, 47, 52, 55, 56, 58, 59, 60, 67, 68, 69, 70, 72, 73, 79
正規化, 26, 29, 37, 38, 47, 49, 74
正規分布, 80
生産工学, 12, 22, 24
生産性, 2, 5, 7, 12, 15, 16, 26, 28, 30, 33, 43, 45, 46, 47, 51, 52, 54, 55, 63, 64, 65, 67, 73, 74, 76, 77, 79, 80, 81, 101
  狭義の-, 51
  広義の-, 54
  広義の −, 54
  広義の −, 55
製品開発, 4, 7, 12, 15, 18, 19, 32, 44, 92, 93, 95, 101
製品企画プロセス, 11, 34, 44
製品検査, 26
セキュリティ, 2, 41, 85, 86, 87, 90, 91, 92, 93, 96, 97, 98, 99
そ
層, 4, 47, 88, 89, 90, 92, 93, 98
総売上, 54, 55
総原価, 51, 54, 55
総資本回転率, 63, 66, 73
組織化, 1, 6, 24, 29, 35, 36, 37, 39, 40, 41, 43, 45, 46, 56, 58, 59, 60, 102
組織的なソフトウェア開発方法, 1, 5, 6, 20
ソフトウェア開発
  回転率, 2, 7, 63, 64, 68, 70, 71, 72, 73, 74, 75, 76, 77, 79, 80, 81, 82, 83, 84, 101
  組織, 1, 2, 3, 4, 5, 6, 9, 10, 11, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 24, 27, 28, 30, 31, 32, 33, 34, 36, 39, 40, 41, 43, 44, 45, 46, 51, 61, 63, 64, 65, 66, 67, 68, 72, 73, 79, 81, 82, 85, 101, 102
  プロジェクト, 1, 2, 3, 6, 7, 10, 11, 14, 16, 20, 26, 27, 28, 30, 31, 37, 38, 39, 40, 43, 45, 46, 47, 48, 49, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 63, 65, 72, 73, 74, 75, 78, 81, 82, 85, 101
  プロセス, 1, 6, 10, 11, 15, 20, 22, 25, 26, 28, 29, 31, 32, 35, 37, 38, 41, 43, 44, 45, 47, 52, 60, 63, 64, 65, 67, 68, 70, 71, 72, 75, 76, 78, 81, 82, 85
  プロセス監査, 34
  プロセス導出モデル, 28
  プロセスモデル, 2, 28, 29, 63
ソフトウェア開発者, 3, 11, 14, 15, 20, 25, 26, 30, 35, 38, 39, 41, 46, 50, 51, 55, 56, 58, 59, 60, 64, 74
ソフトウェア工学, 1, 3, 4, 5, 6, 9, 10, 11, 12, 13, 14, 15, 17, 19, 22, 23, 28, 31, 41, 69, 81, 101
ソフトウェア工場
  日本の一, 4, 15, 26, 28, 64
ソフトウェア構成項目, 56, 68, 69, 70, 71, 76, 78, 79, 82, 83, 84
ソフトウェア受注, 32
ソフトウェア生産技術, 1, 2, 5, 6, 7, 9, 10, 11, 12, 14, 18, 22, 24, 25, 26, 27, 28, 30, 31, 33, 35, 36, 38, 40, 41, 43, 45, 54, 61, 63, 101, 102, 103
  改善, 31, 46, 50
  業務機能, 1, 6, 9, 24, 25, 31, 41, 43, 61, 101
  規律・統制, 31, 39, 59
  組織化, 31, 36, 39, 40, 56
  対象, 1, 6, 9, 11, 24, 25, 27, 30, 31, 35, 38, 39, 41, 43, 61, 81, 101, 102
  モデル, 9, 41, 81
ソフトウェア生産システム, 1, 6, 27, 44, 45, 101
ソフトウェア製品, 7, 26, 32, 34, 43, 44, 48, 50, 51, 52, 53, 54, 55, 58, 60, 101
ソフトウェアフォールト, 34, 38, 47, 48, 53, 56, 64, 65, 74, 79, 82, 83, 84
ソフトウェアプロダクトライン, 19, 33
た
大規模ソフトウェア開発, 1, 2, 3, 5, 11, 12, 43, 44, 45, 61, 63, 64, 73, 81, 84, 93, 101
竹内弘高, 4, 21
多様化, 1, 5, 6, 22, 41
ち
知識管理, 17, 19, 21, 42
知識の場, 30, 31
知的財産権, 1, 2, 6, 7, 21, 22, 30, 31, 39,

40, 41, 46, 85, 101, 102
中間成果物, 26, 27, 69, 70, 81
著作権, 21, 87, 97
て
定着化, 39, 40
定量化, 14, 15, 16, 18, 19, 22, 24, 31, 32, 33, 35, 37, 38, 39, 42, 47, 54, 63, 79, 101
電子メール, 2, 85, 86
と
統計学, 22
ドキュメント, 15, 25, 26, 28, 35, 47, 56, 57, 58, 61, 69, 84
特性要因図, 25
トヨタ生産方式, 4, 13, 20
トレードオフ, 68, 87, 88
な
内部統制, 21, 95
の
野中郁次郎, 4, 21
は
ハードウェア生産, 4, 12, 20, 25, 27
ハードウェア生産技術, 24, 25
ひ
ビジネスプロセス, 85, 88, 90, 91, 93, 95, 99
日立ソフトウェア事業部, 43, 44, 45, 46, 47, 49, 50, 52, 56, 58, 59, 74, 94
標準化, 12, 15, 28, 31, 37, 39, 47, 49, 56, 59, 63, 64
品質管理, 4, 9, 12, 13, 17, 25, 27, 46, 68, 74
ふ
ファイヤウォール, 41, 86
ファンクションポイント, 15, 16, 38, 67
フォーマル型コラボレーション基盤, 94
物理層, 88, 89, 90, 93
部品, 9, 10, 19, 20, 25, 26, 27, 28, 36, 39, 51, 53, 55, 66, 76
部品在庫回転率, 63
プラクティス, 4, 5, 13, 17, 18, 21, 32, 34, 37, 72
フレキシブル型コラボレーション基盤, 99
プログラム, 16, 26, 27, 36, 47, 49, 70
プロジェクト管理, 10, 27, 31, 46, 92, 93, 94, 96
プロジェクトナビゲーションシステム, 60
プロセス成熟度モデル, 4
プロトタイピング, 75, 76, 91, 93
へ
変更管理, 56, 57, 60
ま
マニュアル, 11, 26
マルチプラットフォーム, 50, 51, 52, 55
め
メインフレーム, 50, 60
メトリクス, 1, 7, 17, 19, 32, 33, 34, 35, 37, 38, 47, 50, 51, 54, 55, 56, 63, 64, 65, 66, 68, 70, 74, 76, 81, 101
も
問題定義, 31, 32, 33
り
リーン開発手法, 4, 13
リスク, 21, 29, 37, 40, 41, 46, 68, 72, 86, 87, 88, 89, 90, 91
リトルの法則, 13, 66, 81
ろ
論理層, 88, 89, 90, 93